OS/2

מאפיינים, פונקציות ויישומים



ג'פרי י. קרנץ
אנה מ. מיזל
רוברט ל. וויליאמס
חברים בצוות
הפיתוח של OS/2

מנהל התצוגה
מנהל בסיס הנתונים
המערכת האישית PS/2

ספרי לימוד והכשרה במדעי המחשב
הוצאת הוד־עמי

OS/2

למשה,
דרך צלחה.
8/89

עורך: יצחק עמיהוד

תרגום: חיים פליבה

עריכה לשונית ועיצוב: שרה עמיהוד

OS/2

מאפיינים, פונקציות ויישומים

ג'פרי י. קרנץ
אנה מ. מיזל
רוברט ל. וויליאמס

חברים בצוות הפיתוח של OS/2

נספחים:

מנהל התצוגה
מנהל בסיס הנתונים
המערכת האישית PS/2

ספרי לימוד והכשרה במדעי המחשב
הוצאת הוד־עמי

OS/2

Features, Functions and Applications

Jeffrey I. Krantz
Ann M. Mizell
Robert L. Williams

ISBN 0-471-60709-6

Authorized translation from English language edition
published by John Wiley & Sons Inc.

Hod-Ami Publishers Ltd.
P.O. Box 560, Ramat-Gan 52105
Israel

הוצאת הוד-עמי
לספרי מחשבים בע"מ
ת.ד. 560 רמת-גן

הודפס בישראל
סיון תשמ"ט, 1989
Printed in Israel, 1989

ISBN 965-361-004-x מסת"ב

הספר מוקדש לאנשים

אשר חזונם ויוזמתם ישפיעו באופן משמעותי
על מקומן של תחנות העבודה האינטליגנטיות (ISW)
בעולם עיבוד המידע.

המחברים מודים לאנשים הבאים

Jim Archer, Jay Martinson, Bill Gates, Steve Ballmer

וכן להנהלות של חברות יבמ (IBM) ומיקרוסופט (Microsoft), למעצבים ולמתכננים, למתכנתים ולצוותי הבחינה, אשר עבודת הצוות שלהם ומאמציהם האישיים תרמו לפיתוח של המהדורה הראשונה של OS/2 ולהנחת התשתית למהדורות הבאות.

הערת העורך:

המחברים והמו"לים השתדלו להביא בפני הקורא את המידע העדכני שהיה בידם בעת הכתיבה של הספר.
הספר מותאם לגירסה OS/2 Standard Edition 1.0. הוא כולל עדכונים ותוספות עבור גירסה 1.1 ונושאים נוספים המתייחסים למהדורה המורחבת OS/2 EE (OS/2 Extended Edition).

הנספחים נוספו במהדורה העברית בהרשאה של המו"ל John Wiley ובאדיבות חברת יבמ ישראל.

בעת התרגום השתדלנו להשתמש במונחי המקובלים בעברית. כאשר אין מונח מוסכם, העדפנו להשתמש במושג באנגלית, כפי שהוא מופיע במקור.

סימנים מסחריים - Trademarks

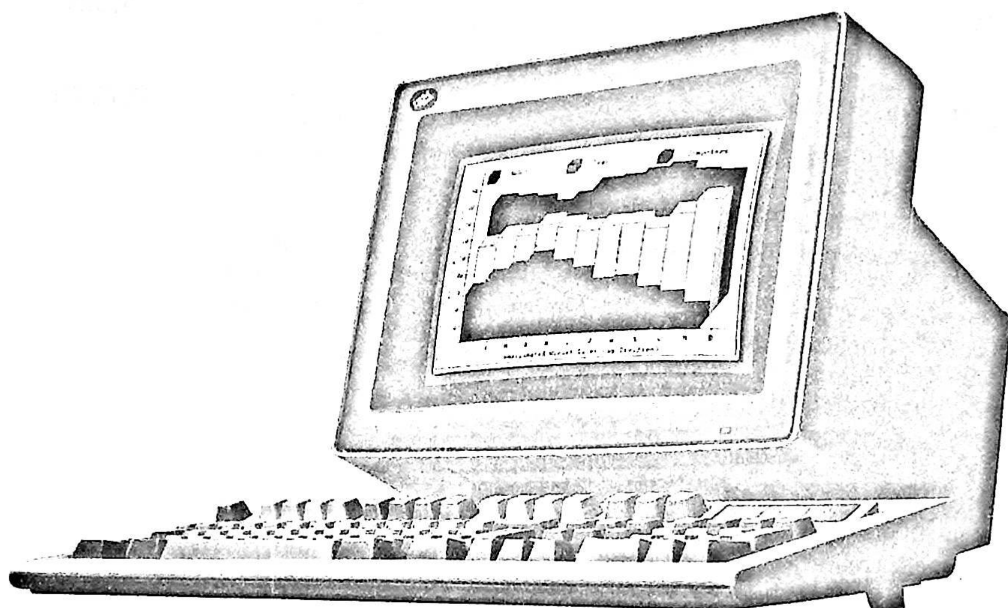
בספר זה הזכרנו סימנים מסחריים רשומים של חברות ושל מוצרים שונים. סימנים נוספים צוינו בגוף הספר.

יבמ, IBM
< סימנים מסחריים של International Business Machines Corporation
מיקרוסופט, Microsoft
< סימנים מסחריים של Microsoft Corporation
אינטל, Intel
< סימנים מסחריים של Intel Corporation

הסימנים המסחריים הבאים הם של יבמ (IBM),
International Business Machines Corporation

- Operating System/2; OS/2
- Personal System/2; PS/2
- PC XT
- Personal Computer AT; AT
- Presentation Manager
- Personal Computer XT Model 286
- Micro Channel
- AS/400

תוכן העניינים



הקדמה

ההחלטה לכתוב ספר זה היתה החלטה קלה. מאחר ואנו, הכותבים, אנשי מפתח בקבוצת הפיתוח של מערכת ההפעלה OS/2, מצאנו את עצמנו עם כמות אדירה של ידע מעמיק ורחב יריעה על המוצר. הרגשנו שהדרך היחידה שבה נוכל לנצל את הידע הזה בצורה טובה היא לכתוב ספר. לצערנו, קל להגיד אך קשה לבצע. כמות המידע על המוצר שהיתה במוחנו יכולה היתה למלא מספר כרכים. לכן, רצינו לכתוב ספר שהקהל שלנו יוכל לעכל. לרוב האנשים יש עבודה אמיתית לעשות, אך רק כמות זמן מוגדרת לעשות זאת. השאלה הקשה ביותר שנאבקנו אתה היתה איזה סוג ספר אנו רוצים לכתוב. רצינו שהתועלת שתצמח לך כתוצאה מקריאת ספר זה תהיה מקסימלית. עשינו זאת על-ידי כיסוי סלקטיבי של החומר הקיים על OS/2.

הספר שאתה מחזיק בידך הוא תוצאה של מאמצינו. הספר מכסה לפי הבנתנו את האספקטים החשובים ביותר של Operating System/2 גירסה 1.0 (הערה: בנספחים יש השלמות לגירסה OS/2 EE). אם אתה חבר באחת מהקבוצות הבאות, תמצא שספר זה יועיל לך מאוד:

- משתמשי DOS על מחשבי PC.
- אנשים הסקרנים לגבי הישימות של OS/2 לבעיותיהם.
- מנתחי מערכות.
- מעצבי מערכות.
- מתכנני מערכות.
- מפתחי יישומים.

אינך צריך להיות בקי בפרטים של מערכת ההפעלה DOS למחשבי IBM-PC ותואמיהם, או במיקרו-מעבד אינטל 80286 כדי להפיק תועלת מספר זה. הבאנו בספר חומר רקע כדי שתוכל להבין את מערכת ההפעלה OS/2 ואת הישימות שלה. מכיון שסביבת OS/2 עולה בתיחכומה על הסביבה הפשוטה במחשב האישי וגם מספקת פונקציות מערכת רבות ומתוחכמות יותר, הסברנו עקרונות של מערכת הפעלה שנשתמש בהן בספר זה.

הספר מסביר מדוע האפשרויות הגלומות ב-OS/2 חשובות לך, ומדוע הבעיות ש-OS/2 פותרת חשובות לך. השתמשנו, היכן שניתן, באפשרויות של DOS כנקודת ייחוס.

איך לקרוא ספר זה

כמות המידע והנתונים הקיימים על OS/2 הם רבים. בחרנו על כן, לתאר את מה שנראה לנו כמאפיינים החשובים ביותר של המוצר, והפחתנו את כמות הפרטים על כל מאפיין לרמה הניתנת לקליטה ולהבנה ללא מאמץ לימוד מיוחד. לאחר שתקרא ספר זה על האפשרויות ש-OS/2 מספקת, יהיה השימוש בפרסומים הטכניים הרבים הקיימים לגבי המוצר משימה קלה ויצירתית. חילקנו את התיאורים של האפשרויות השונות ב-OS/2 לשלוש רמות כלליות של פירוט, אשר תאפשרנה לך להיות גמיש בקריאת הספר.

תחילה, תיארנו רבות מהאפשרויות של OS/2 בצורה מקוצרת. כך תבין את התפישה, את התועלת שניתן להפיק מן השירותים השונים של המערכת, ואיך ניתן ליישם אותם לבעיות שאתה צריך לפתור. ברמה הבאה ירדנו עמוק יותר לפרטים לגבי כל שירות (כמו ניהול זיכרון למשל). עשינו זאת על-ידי תיאור המישקים המתאימים המאפשרים תכנות של יישומים. ברמה זו הצגנו את הפרטים החשובים ביותר של המישקים, כדי שתוכל לראות איך OS/2 מצליחה לבצע את מה שהוגדר קודם כאפשרות בלבד. רמת הפירוט הזו חשובה במיוחד, כי היא מאפשרת לך לראות איך ניתן לבנות יישום מסוים, אשר יוכל לנצל בצורה מקסימלית את האפשרויות הגלומות ב-OS/2. סיפקנו רמה שלישית של פירוט לאלה הרוצים לראות איך לנצל את האפשרויות ש-OS/2 מציעה על-ידי שימוש בשפת מכונה. כללנו שני פרקים שמציגים את האפשרויות החשובות ש-OS/2 מספקת, בעזרת דוגמאות של תכניות הניתנות להידור והרצה.

התיאורים של OS/2 המסודרים בשלוש רמות שונות של פירוט, מאפשרים לך לבחור את הדרך הנוחה לך בקריאת הספר. אם אינך מתעניין בדוגמאות התכנות, אתה יכול לדלג על פרקים 4 ו-7 מבלי לפגוע בהבנת הספר. עם זאת, יכול להיות שתעדיף לפנות ישר לדוגמת תכנית לתמיכה בזיכרון יתר (memory overcommitment) בפרק 4, לאחר שקראת בפרק 2 את הדיון המקדים על אפשרויות ניהול הזיכרון. לאחר שעברת על דוגמת התכנות, אתה יכול לחזור לפרק 2 ולקרוא על המישקים המאפשרים את ניהול הזיכרון. אנו מציעים לך לקרוא ספר זה מתחילתו עד סופו, כדי לקבל את התמונה הרחבה והמפורטת ביותר על OS/2. הספר נבנה במיוחד כדי שתוכל לבחור את הדרך הנוחה ביותר לעיון בו, בהסתמך על כמות הפרטים וסדר ההצגה שאתה מעדיף.

מבוא

סקירה היסטורית

במטרה להבין להיכן ה-OS/2 מובילה את המיחשוב האישי, הרגשנו שחשוב להיזכר היכן היינו. הדיון הבא הינו שילוב של נקודות המבט האישיות של הכותבים.

המחצית הראשונה של שנות ה-70

בתקופה זו נוצר המפגש הראשון בינינו לבין עולם המיחשוב. זה קרה בסביבה אקדמית, אשר אופיינה כדלקמן:

- מרכז חישוב "גדול" (פיסית) וריכוזי.
- מכונות לניקוב כרטיסים.
- זרם עבודות אצווה (Batch) שהוזנו למחשב בתיווכו של המפעיל.

כיום, אפשר לתאר את הסביבה המוקדמת הזאת כפרימיטיבית מנקודת המבט של פשטות השימוש ופריון העבודה. הכנת הנתונים והתכנות עצמו נעשו באמצעות מכונות ניקוב. אנשים איחסנו את כל המידע והתכניות שלהם בחבילות גדולות של כרטיסים מנוקבים. הגברת פריון העבודה בסביבה זו נעשתה בעזרת תוף ניקוב או בעזרת מונה טורים סיפרתי על מכונת הניקוב שלך (אם היה לך מזל). אם היית חכם מספיק היית מנקב מספרים עוקבים על כל כרטיס בחבילה. ואז, כאשר המפעיל מחליט לזרוק את החבילה שלך, או כאשר קורא הכרטיסים מחליט שהוא רעב, היתה לך אפשרות להתאושש. שטח על דיסק פרטי נחשב למשאב יקר, ולא היה זמין בכמות מספקת למשתמש הממוצע.

כל חבילות הכרטיסים של המשתמשים נאספו יחד על-ידי מפעיל המערכת והוזנו לתוך קורא הכרטיסים. תהליך זה ידוע בשם עיבוד באצווה (batch processing). בתהליך זה תכניות המשתמשים הוזנו בקבוצות ללא התערבות המשתמש. לאחר זמן מה, בד"כ חצי יום או יותר, חיכית בתור כדי לקבל תדפיס של תוצאות ההידור או ההרצה של התכנית שלך. אם התכנית שלך לא פעלה כשורה, היית צריך להתחיל את כל התהליך הארוך הזה מחדש. ברור, שהתהליך של לימוד מטעויות שלך היה קשה ואיטי. המחיר של טעות אחת יכול היה להגיע עד כדי בזבוז של יום עבודה שלם.

בנוסף לכך, מחיר המיחשוב היה גבוה מאוד. סעיפים רבים היו צריכים להלקח בחשבון לצורך חיוב כמו:

- כל מאית ואלפית שניה של זמן עיבוד במחשב.
- כל פעולת קלט או פלט שהמחשב ביצע.
- כל יחידת אחסון שנעשה בה שימוש במתקני האחסון החיצוניים.
- כל פעולה שהמפעיל היה צריך לבצע כדי לאפשר לתכנית שלך לרוץ.
- כל שורה או דף של פלט מודפס שקיבלת.

בנוסף לכך, כמות משאבי המחשב שתכנית יכולה היתה לצרוך היתה מוגבלת. לדוגמה, בסביבה של מחיצת זיכרון קבועה, מנהל המערכת היה צריך לחלק למספר קטעים תכנית הצורכת יותר מ-256KB זיכרון.

בסביבה אקדמית זו, האתגר הגדול ביותר של כולם היה להשיג עזרה כאשר היה צורך. ככלל, מפעיל המערכת היה האדם היחיד שהמשתמשים (סטודנטים) ראו אותו באופן קבוע. באופן כללי, מפעיל המחשב לא היה האדם הנכון לפנות אליו בנושא של עזרה טכנית. היית נותן למפעיל את הכרטיסים שלך, והוא (או היא) היו מחזירים לך את הכרטיסים והתדפיסים לאחר ביצוע. אם היית צריך עזרה, היית צריך להתייעץ עם תכניתן המערכת. תכניתן המערכת היה מומחה, אדם היה בעל ידע ונסיון שאיפשרו לו לעשות דברים לא שיגרתיים.

במבט לאחור על סביבת המיחשוב בתקופה שהיינו באוניברסיטה, אנו יכולים לאפיין אותה בדרך הבאה:

- יקרה.
- זמן סבב ארוך.
- משתמשים מופרדים ממערכת המחשב.
- משאבי מחשב מעטים.
- תהליך לימוד קשה ואיטי.
- קשה לבצע ניסויים.
- קשה לקבל עזרה.

מאמצע עד סוף שנות ה-70

בתקופה זו, היינו עדים למספר שינויים בסיסיים שחלו במרכזי המחשב הגדולים והריכוזיים. מסופים הידברותיים (Interactive) עם מסכי CRT הותקנו בכמויות גדולות. ההתקנות האלו איפשרו למשתמש הסופי להידבר ישירות עם המחשב.

משתמש הכניס נתונים דרך המקלדת ויכול היה לבדוק אותם ישירות על המסך, או לראות את הפלט של התכנית לפני בקשת תדפיס. אפשרות חדשה זו האיצה באופן משמעותי את תהליך המשוב במקרה של טעויות. אפשר היה להריץ תכניות משתמש הן באצווה והן במקוון, עם משוב בזמן אמיתי על המסך. כמובן, שלמדת מהר להריץ באצווה תכניות שצרכו כמויות גדולות של משאבי מחשב.

מכיון שמשאבי מרכז המחשבים היו עדיין יקרים, היה צורך לאפשר למספר גדול של משתמשים להשתמש במחשב בו-זמנית. במספר מתקנים התפתו להעמיס את המחשב מעבר למה שאפשר, במטרה לשמור על יחס נמוך של הוצאות/משתמשים. אם המערכת הועמסה מעבר ליכולתה, הדבר גרם לכך שלפעמים לא היית יכול לקבל שירות, כי יותר מדי אנשים השתמשו במערכת באותו רגע. עומס יתר גרם לכך שמשך הזמן שעבר בין הבקשות לשירותי מערכת לבין תשובת המערכת על המסך, התארך בצורה בלתי נסבלת.

בהתחלה, המסופים (CRT) היו יקרים יחסית ולכן, בד"כ לא היה לאנשים מסוף אישי על שולחנם. כתוצאה מבעיות של זמן תגובה הנובעות ממערכות שפועלות בעומס יתר וכתוצאה מבעיה של מציאת מסוף פנוי, לא גדל פריון העבודה בצורה משמעותית כפי שניתן היה לצפות כתוצאה מעבודה בסביבת מיחשוב הידברותית.

אולם, שנות ה-70 המאוחרות סימנו את תחילת התקופה של הידברות אישית בין המשתמש לבין המחשב. מאחר וההידברות הינה תהליך של אחד לאחד, הזמן שדרוש לבצע ניסויים במערכת מתקצר באופן משמעותי, ולכן המשתמש מקבל תמריץ לשפר את ביצועיו במערכת.

מסוף שנות ה-70 עד אמצע שנות ה-80

אפשר לאפיין תקופה זו כזמן שבו המחשב הגיע אל האדם הממוצע, או כתקופת ההתפשטות של המיחשוב האישי. גם סביבת המחשבים הגדולים השתפרה במשך תקופה זו בצורה כזו, שמשאבי מרכז המחשבים הפכו להיות חזקים יותר, גמישים וזולים יותר. המסופים התחילו להופיע על כל שולחן, וחדרי מסופים החלו להעלם. אבל זו היתה התפתחות טבעית של העשור הקודם. הסיפור האמיתי הוא המהפכה שחולל המיחשוב האישי.

כאשר אפשר לרכוש מערכת מחשב שלמה בעלות של אחוז קטן מתוך ההכנסה השנתית של עובד, והיא יכולה לעמוד בפינת השולחן שלו, אזי אפשר לצפות שהדבר יגרום למהפכה במאפיינים של המיחשוב. מחשבים חדשים אלה היו כה זולים שלא היה צורך להתחלק בהם עם אנשים רבים בו-זמנית, ואפשר היה לתכנן אותם כמערכות של משתמש יחיד. התפתחות זו הפחיתה בצורה משמעותית את המורכבות של מערכת ההפעלה שנדרשה כדי לנהל את משאבי המערכת. אדם היה יכול לשלוט על מה שהמחשב שלו, או שלה, נדרש לעשות. רמה זו איפשרה לאנשים לבצע שיקול של עומס העבודה על המחשב האישי מול זמן התגובה הנדרש.

מספר האנשים שהיתה להם גישה למחשבים גדל במהירות. אפילו בגני הילדים ניתנה לילדים האפשרות לגשת ולעבוד עם מחשב אישי. יש מספר סיבות שבגללן התפשט הידע במחשבים בצורה מהירה בחברה שלנו:

- היחס של אחד לאחד (אדם-מכונה) האפשרי במחשבים האישיים.
- האופי הפשוט של היחסים.
- היכולת לקבל משוב מיידי על מה שאתה עושה.
- גישה קלה למחשבים אישיים.

אין להבין מההתפתחויות אלו שהתועלת והצורך במערכות מחשב גדולות נעלמו. למעשה, הן גדלו. המחשבים האישיים החדשים לא החליפו את התפקיד או הצורך במערכות מחשב גדולות. בפועל, הם יצרו בעצמם צורך, שנדחף על-ידי קבוצות חדשות של אנשים ושווקים. במקביל, מערכות המחשב הגדולות הפכו להיות חזקות יותר ויצרו אפשרויות חדשות וכל זאת תוך שיפור המחיר ביחס לביצוע.

המחשב האישי IBM PC ומערכת ההפעלה DOS

עם כניסתם לשוק של מחשבי IBM PC ומערכת ההפעלה DOS בשנות ה-80 המוקדמות, יצרה IBM את מה שנועד להיות אחת מסביבות המיחשוב האישיות הפופולריות ביותר. רוב האנשים מייחסים את ההצלחה של IBM PC לא רק למאפייני החומרה של המערכת, אלא גם לכמות הגדולה של תוכנות יישום שהיו זמינות לשימוש במחשב תוך זמן קצר, הן מצד יבמ והן מצד משווקים אחרים.

תוכנת היישום והסביבה שבה היא הורצה, עשו יותר מאשר פתרון של בעיות אמיתיות שהיו למספר גדול של אנשים ועסקים. פשטות השימוש בסביבת התוכנה הקלה על הגישה למספר גדול של אנשים. המאפיינים של תוכנת היישום בתקופה זו נגזרו ממגבלות מערכת ההפעלה והחומרה שהיא רצה עליה.

אלה המאפיינים של הסביבה הזאת:

- מערכת של משתמש יחיד.
- זיכרון פנימי של 640KB (RAM).
- מערכת חד-משימתית (ביצוע פעולות בצורה סידרתית).
- יישום אחד המופעל בו-זמנית.

נראה שסביבה זו ענתה על צרכי שוק יוצאי דופן, ופתרה בעיות למספר גדול של עסקים ואנשים. אז לאן אנו מתקדמים מכאן? ומדוע בכלל אנו רוצים להתקדם מכאן?



פרק 1

מבט אל OS/2

היתרונות העיקריים של הגירסה הראשונה של מערכת ההפעלה OS/2 ניתנים לחלוקה למספר קטגוריות:

- שוברת את מחסום 640KB של הזיכרון הפנימי.
- מספקת תמיכה לביצוע רב משימות (Multitasking).
- מספקת מישק לתכניות יישום (API).
- מספקת סביבה המאפשרת הרצת מספר יישומים במקביל.
- אפשרות ביצוע של יישומי DOS.
- מספקת מישק עקבי למשתמש.

מדוע OS/2?

מה אומרים לך דברים אלה? מדוע אתה צריך להתעניין בדברים כמו **ריבוי משימות** (Multitasking) **ומישק לתכניות יישום** (Application Programming Interface - API)? מדוע סביר שסביבת DOS הנוכחית אינה עונה על צרכיך העכשוויים והעתידיים? כדי שנתחיל להבין את התשובות, הבה נסתכל על מספר סביבות וצרכים של משתמשים שיכולים לגרום לכך שיתרונות אלה ייראו חשובים:

- תחנת עבודה עם אפשרויות רבות המורכבת מטכנולוגיות מתקדמות של חומרה.
- סביבות חדשות.
- משתמשים מיצו את הפוטנציאל הקיים במחשבים אישיים עם מערכת ההפעלה הנוכחית.

תחנות עבודה בעלות טכנולוגיות חומרה מתקדמות

הטכנולוגיה משתפרת בצעדי ענק. לפני עשר שנים מערכות שהיו להן 48KB של זיכרון פנימי (RAM) ותצוגה של 16 שורות ו-64 טורים נחשבו למחשבים אישיים פופולריים ושימושיים (ראה גם סעיף בפרק זה העוסק בסקירה של המיקרו-מעבד 80286). מחשבי IBM PS/2 החדשים מתחילים עם 1MB של זיכרון פנימי על הלוח הראשי בדגם 50, ובדגם 80 אפשר להגיע עד 4MB על הלוח הראשי, ללא כרטיסי הרחבה. שיפור זה ביחס עלות/ביצוע בטכנולוגיית החומרה מגדיל את הכדאיות שברכישתה כפי שנהוג מזה זמן רב בענף המחשבים. רמת התיפקוד הנוכחית של DOS אינה מסוגלת לנצל בצורה מלאה את השיפורים בחומרה.

השיפורים המשמעותיים ביותר בחומרה הם במהירות **המעבד** עצמו, בפעולות שהוא מאפשר (מאינטל 8088 לאינטל 80286 עד לאינטל 80386) ובכמות **הזיכרון** הפנימי שמערכות אלו יכולות להכיל. בנוסף לכך, יש אפשרות לבצע עיבודים גרפיים ברמה דומה למערכות ייעודיות לגרפיקה.

DOS מספקת רמת תיפקוד המנצלת בצורה מלאה את המיקרו-מעבד אינטל 8088. במחשבי IBM PC/AT ו-IBM PS/2 דגמים 50 ו-60 הוכנס המיקרו-מעבד אינטל 80286 במקום 8088 (יש מודלים נוספים שבהם מותקן 80286). ב-IBM PS/2 דגם 80 הוכנס המיקרו-מעבד אינטל 80386. ב-80386/80286 ישנו מצב הנקרא **מצב אמיתי** (real mode), אשר תואם ל-8088. זהו מצב שבו ה-DOS פועל כאשר משתמשים ב-80286 או 80386 במחשבי IBM PC/AT ו-IBM PS/2 דגם 50 ומעלה. במצב אמיתי, אף לא אחד מהמאפיינים החדשים של ה-80286 בא לידי ביטוי. ונכון, איכפת לנו שנשתמש במאפיינים החדשים האלה. לדוגמה, ה-80286 מספק אפשרות להגן על מערכת ההפעלה ועל קבצי נתונים מפני תכניות יישום ולהגן על יישומים שונים זה מזה; זהו **מצב מוגן** (protect mode). ה-80286 גם מאפשר תמיכה בזיכרון בפועל ותמיכה בזיכרון אמיתי הגדול מ-1MB. OS/2 עצמה מנצלת את המאפיינים של ה-80286 ומאפשרת לתכניות יישום לנצל אותם. אנו רואים שחייבים לעבור ל-OS/2, אם רוצים לנצל את מלוא היתרונות שהטכנולוגיה החדשה של המיקרו-מעבדים מציעה.

ככל שהזמן עובר, המיקרו-מעבדים והמערכות שהם נמצאים בתוכם הופכים להיות מהירים יותר ויותר. כדי להגיע לניצול מלא של הפוטנציאל הטמון במערכת המיחשוב, מערכת ההפעלה צריכה לתמוך בביצוע של יותר ממשימה אחת בו-זמנית. זאת אומרת, שמערכת ההפעלה צריכה לאפשר לחלקים שונים של עבודה, או יישומים, להופיע כאילו הם מבוצעים בו-זמנית. לדוגמה, בזמן שחלק אחד של העבודה צריך לחכות בגלל נתונים המגיעים מהתקן קלט/פלט כלשהו, המערכת יכולה לעבד חלק אחר של העבודה. ככל שהמערכות הופכות לחזקות יותר, אפשרות זו הופכת לחשובה יותר. DOS אינה מספקת מנגנון אשר נותן שירות זה, אך OS/2 כן עושה זאת.

מכיון ש-DOS תומכת רק ב-8088, או במצב האמיתי של 80286, היא יכולה לתמוך ביישום בגודל מקסימלי של 640KB במחשבי IBM PC. מרחב הכתובות בין 640KB לבין 1MB מנוצל על-ידי יחידות קלט/פלט ממופי-זיכרון ועל-ידי פקודות ה-ROM BIOS. אין ל-DOS שום דרך לבצע תכנית יישום הגדולה מ-640KB. יישומים אינם יכולים לגשת ישירות לנתונים שלהם, אם הנתונים מאורגנים בקבצים העולים על מחסום ה-640KB. לכן, יישומים הרצים בסביבת DOS אינם יכולים לבצע אופטימיזציה של השימוש בכל הזיכרון שקיים במערכות המחשב האישיות החדשות. מכל מקום, OS/2 מאפשרת ליישומים לרוץ **במצב מוגן** (protect mode) שהוא מצב הפעולה המלא של ה-80286. יישומים אלה יכולים להתבצע ביותר מ-1MB זיכרון ויכולים להתייחס ישירות לנתונים המאורגנים בקבצים הגדולים מ-1MB זיכרון.

ככל שהיישומים הופכים ליותר מתוחכמים, הם צורכים יותר מקום בזיכרון, אשר בו צריכות להתבצע ההוראות שלהם. בנוסף לכך, ככל שכמות הנתונים שצריכה להיות מעובדת בזמן נתון גדלה, כך הופך לקריטי, מבחינת תגובת המערכת, השטח הנוסף בזיכרון שנועד לנתוני היישום. ככל שיישומים מנצלים יותר את התצוגה הגרפית המתוחכמת (רזולוציה גבוהה וצבעים רבים), כך גדלה כמות השטח שיש להקצות לנתונים הדרושים ליישום. מכאן ברור מדוע אנו רוצים לנצל את כל מיליוני הבתים אשר מציע הזיכרון במערכות החומרה החדשות.

נאמר כבר שתמונה שווה אלף מלים. קל יותר להשתמש במישק גרפי, אשר יכול לתמוך בסביבה יישומית מתוחכמת טוב יותר מאשר מישק-משתמש בשיטה המקובלת. ככל שהמשתמש מטפל במשימות רבות יותר בו-זמנית, כך גובר הצורך לראות אותן בצורה של **חלונות** (windowing), ואת זה מספק אותו מישק גרפי. בסביבת חלונות, יישומים שונים תופסים חלקים שונים של המסך. גירסה 1.0 של OS/2 מספקת מישק-משתמש מבוסס תווים לטיפול במסך שלם. גירסה 1.1 והגירסה OS/2 EE מאפשרות ליישומים לנצל בצורה מלאה את מישק המשתמש המבוסס על גרפיקת חלונות בעזרת **מנהל התצוגה** (Presentation Manager).

סוגי החומרה של IBM הנתמכים על-ידי OS/2

OS/2 אינה פועלת במחשבים אישיים שמותקנים בהם מיקרו-מעבדים מסוג 8086 או 8088. זאת אומרת, ש-OS/2 אינה רצה על מערכות כמו IBM PC, PC XT, PS/2 דגמים 25 ו-30 ותואמיהם.

בזמן כתיבת ספר זה, OS/2 תומך בסוגי החומרה הבאים של יבמ:

- מחשבי IBM PC/AT (5170) דגמים 068 עם דיסק קשיח (מקביל לדגם 099), 099, 239, 319, ו-339.
- מחשבי IBM PC/XT דגם 286 (5162).
- מחשבי IBM PS/2 דגמים 50, 60, 70 ו-80.

ה-80386 ב-PS/2 דגם 70 ו-80 נתמך בדומה ל-80286 (כך המצב כעת).

לא כל מתאם הקיים לכל דגם נתמך על-ידי OS/2. OS/2 דורשת לפחות כונן דיסקטים אחד בעל קיבולת גבוהה, דיסק קשיח, מקלדת, ומתאם תצוגה עם המסך הקשור אליו. OS/2 דורשת לפחות 1.5MB של RAM כאשר רק OS/2 מורצת. אם משתמשים גם ב-DOS, OS/2 דורשת לפחות 2MB של RAM. סביבת הביצוע של DOS וסביבת הביצוע של OS/2 מתוארות בהמשכו של פרק זה.

סביבות עבודה חדשות

המבוא לספר זה מציג שני סוגים נפרדים של סביבות מיחשוב שהיו לנו עד היום: סביבת המחשב האישי וסביבת המערכת הגדולה. מחשבים אישיים תוכננו כך שישתמשו וישלטו בהם משולחן של מישהו. הם מאפשרים לאנשים להתאים את עיבוד הנתונים שלהם לפי צרכיהם. אבל העוצמה של המחשבים האישיים מוגבלת. כמו כן, הכמות והטווח של הנתונים המאוחסנים במחשב אישי מוגבלים. סביבת המערכות הגדולות מנוצלת לצורך יישומים הדורשים כמויות גדולות של כוח מיחשוב ונתונים שמתחלקים בהם משתמשים רבים. מערכת גדולה יכולה להיות מבוססת על משתמשים רבים המפוזרים באתרים גיאוגרפיים שונים ומנצלים כוח מיחשוב של מחשב אחד או יותר. הנתונים שאליהם יכולים המשתמשים לגשת יכולים להיות מרוכזים במחשב אחד ענק, או מפוזרים במספר מחשבים קטנים יותר. אפשר לתחום את סביבת המערכת הגדולה באתר פיסי אחד.

אולם במציאות, לא כל היישומים או הפתרונות מתאימים בדיוק לאחת משתי הקטגוריות שתוארו לעיל. לדוגמה, כלכלן העוסק בניתוח פיננסי בחברה גדולה לא יוכל לאחסן את כל הנתונים הדרושים לעבודתו במחשב אישי. אולם, אותו כלכלן יעדיף להשתמש במחשב האישי מכיון שהשימוש בו מאפשר לו הרבה יותר גמישות בתהליכי העיבוד, הניתוח והטיפול בנתונים. מערכות גדולות

נוטות לאפשר פחות גמישות, אבל יכולות לטפל בכל הנתונים הפוטנציאליים שאנשים יצטרכו כדי לעשות את עבודתם. דוגמה אחרת יכולה להיות יישום הדורש זמני תגובה מהירים מאוד בתנאים מסוימים, ותחת תנאים אחרים הוא דורש עיבוד מורכב באמצעות מערכת גדולה. התכנון הטוב ביותר לשתי הדוגמאות הוא הצעד הלוגי הבא בהתפתחות של המחשבים האישיים והמערכות הגדולות, אשר נקרא **עיבוד משותף** (Cooperative Processing). בעיבוד משותף, תחנת העבודה המקומית (מחשב אישי) והמערכת הגדולה מבצעות את העבודות המתאימות להן ביותר, כל אחת בתחומה.

בסביבת העיבוד המשותף, תכנית יישום במחשב אישי תצטרך לבצע יותר ממשימה אחת בו-זמנית. מערכת ההפעלה המטפלת בתכניות יישום במחשב האישי חייבת על-כן:

(1) לתמוך בציוד מתוחכם לתקשורת בזמן אמיתי,
(2) לאפשר גישה לתקשורת לוגית מסובכת ולפרוטוקולים של רשתות בנוסף לגישה לארכיטקטורות להעברת הנתונים.

תמיכה זו דורשת מספר רב של תוכנות ייעודיות לתקשורת, אשר לא תכנסנה ב-640KB ביחד עם תכנית יישום מתוחכמת. סביבת המערכות צריכה לתמוך בזמן אמיתי בחומרה מבוקרת פסקים (Interrupt Driven Hardware) ובמספר משימות בו-זמנית. רמה זו של תיפקוד סופקה כבר ב-DOS בעזרת תכניות מיוחדות, עבור מערכות ויישומים מיוחדים. בסביבת OS/2 התמיכה יכולה להיות בצורה כללית שתאפשר בסיס מודולרי לצמיחה ולהרחבה של תכניות יישום ותת-מערכות בכל הכיוונים האפשריים. ב-OS/2, הגורם העיקרי שמאפשר את המודולריות הוא המישק לקישור דינמי של תכניות יישום - API, שנדון בו בהמשך.

סביבת OS/2 מאפשרת תכנון ופיתוח בצורה כללית של יישומים או מערכות, עם פעילות רבה מאוד של עיבוד משותף, מבלי שהמתכנן יצטרך להבין את כל הדרכים שבהם ניתן להשתמש במערכת לצורך תכניות יישום נוספות בעתיד. תכונה זו מספקת אפשרות לא רק למשווקים עצמאיים הרוצים לספק תת-מערכות, אלא גם לארגונים הרוצים להחזיר את ההשקעה שלהם בעזרת הרחבות מתוחכמות של תוכנת המערכת. OS/2 תומכת בצורה מפורשת במספר יישומים העובדים בו-זמנית. אם נוסיף לכך שמישק המשתמש הוא עקבי, נקבל שהפוטנציאל לניצול השיפורים במערכות, בין אם אלה שסופקו על-ידי היצרן ובין אם אלה שפותחו בנפרד, הוא גבוה. יישומים יוכלו להכתב כעת לסביבת מערכות ולארכיטקטורה בדרך כזו, שהם יפעלו אפילו אם נוספו יישומים נוספים למערכת או אם השתנו תת-המערכות שמתחתם. תת-מערכות יכולות להשתנות מכיון שהסביבה החיצונית הפיסית של תחנת העבודה משתנה, או מכיון שסוג החומרה הנתמך על-ידן משתנה. המשתמש יוכל לנצל סביבה מרובת יישומים זו, כי מישק המשתמש לא רק מאפשר ליישומים שונים להתקיים במקביל, אלא גם מאפשר למשתמש להריץ אותם בו-זמנית.

מיצוי הפוטנציאל של המחשבים האישיים

אנו חיים בתקופה של ציפיות גבוהות. מה שהרשים אותנו בעבר והדהים אנשים רבים לפני חמש שנים הפך כבר לשיגרה. אנשים מחכים עכשיו לצעד הבא בטכנולוגיית המחשב האישי שיאפשר להם לבצע את עבודתם בצורה טובה יותר ולהגביר את יכולת התחרות שלהם בשוק.

כפי שראינו, הצעד הבא אינו יכול להתבצע רק בעזרת טכנולוגיית חומרה חדשה. שאלה חשובה היא, איך המשתמש ינצל את טכנולוגיית החומרה הקיימת ואת זאת שתכנס בעתיד. כדי לענות על שאלה זו, נצטרך להסתכל שוב על סביבת מערכות ההפעלה.

משתמשים מצפים לביצוע של משימות רבות יותר במטרה להגביר את פריון העבודה. אחת הדרכים להשיג זאת היא לאפשר ליותר יישומים לרוץ בו-זמנית על מחשב אישי. אולם, חשוב שהשימוש בפונקציה זו יהיה כללי, כך שסוגי היישומים שתרצה להריץ במקביל על המחשב האישי יוכלו אומנם לעשות זאת זה בצד זה. בנוסף לכך, סביבה זו חייבת להיות מוצגת למשתמש בצורה עקבית וקלה לשימוש. ככל שהמערכת הופכת לחזקה יותר, כך פשטות ההפעלה שלה הופכת לחשובה יותר. אם המערכת תהיה מסובכת מדי לשימוש, אנשים יחזרו למערכות שמאפשרות פחות גמישות בהפעלה. אנשים לא יאמינו שיש להם את החופש להפעלת מערכת המחשב האישית בדיוק לפי צרכיהם. יהיה קשה להראות שאחד המאפיינים הבסיסיים של המחשב האישי ממשיך להתקיים.

משתמשים מצפים לא רק לאפשרות להריץ מספר יישומים, אלא גם לאפשרות להריץ יישומים מתוחכמים וחזקים יותר. כפי שנראה, העוצמה של OS/2 מאפשרת ליישומים מתוחכמים יותר להכתב בדרך כזו, שהם יוכלו לפעול זה בצד זה בסביבה מרובת יישומים. יישומים חזקים יותר נכתבו בסביבת DOS תוך שימוש בפונקציות הנמצאות מעבר לסביבה הטבעית של DOS. עובדה זו אינה מאפשרת להם לעבוד בצורה טובה בסביבה הכוללת יישומים אחרים, שיתכן וינסו להשתמש גם הם בפונקציות הנמצאות מעבר לסביבה הטבעית של DOS.

העוצמה של OS/2

מהם המאפיינים הבולטים ביותר ב-OS/2? ברשימה הבאה נראה ריכוז שלהם:

1. סביבת הביצוע של DOS יכולה להתקיים במקביל לסביבת הביצוע של OS/2.
2. זיכרון אמיתי גדול וזיכרון בפועל.
3. תמיכה בזיכרון-יתר.
4. סביבה רבת יישומים.
5. סביבה רבת-משימות.
6. תקשורת בין תהליכים.
7. מישק קישור דינמי.
8. מערכת קבצים תואמת ל-DOS.
9. חומרה מבוקרת פסקים.
10. תת-מערכות לקלט/פלט כמו תצוגה, מקלדת ועכבר.
11. תמיכה בשפות לאומיות ואמצעים לטיפול בהודעות.

סביבת הביצוע של DOS וסביבת הביצוע של OS/2

סביבת הביצוע החדשה והחזקה יותר של OS/2 דורשת שיישומים ייכתבו בהתאם למערכת חדשה של כללים, השונה מהכללים הנהוגים בסביבת DOS הישנה. לכן, יישומי DOS שאנו רגילים להם אינם יכולים לרוץ בסביבת הביצוע החדשה של OS/2. סביבת ביצוע של DOS סופקה יחד עם OS/2, כדי שאפשר יהיה להריץ בה את יישומי DOS הישנים, עד שייכתבו מחדש ל-OS/2. סביבה זו דומה ל-DOS

גירסה 3.3 וההתייחסות אליה בהמשך תהיה כ"סביבת הביצוע של DOS". משתמשים יכולים לעבור מסביבת הביצוע של DOS לסביבת הביצוע של OS/2, ולהנות מהעוצמה של OS/2. יישומים הרצים בסביבת DOS חייבים להיות מותאמים לכללים של DOS גירסה 3.3, בעוד שיישומי ה-OS/2 חייבים להיות מותאמים לכללים החדשים, המותאמים לעוצמה החדשה.

יישום ה-DOS משתלט על המסך והמקלדת כאשר המשתמש עובד בסביבת ביצוע זו. המשתמש רואה את כל מה שיישום ה-DOS עושה על המסך. יישום ה-DOS יקבל את ההקשות שהמשתמש יקיש, פרט לצירופי מקשים מיוחדים המשמשים כמקשים "חמים" של המערכת. מקשים "חמים" אלה מאפשרים למשתמש לעשות דברים שונים, כמו להשעות את סביבת הביצוע של DOS ולאפשר לסביבת הביצוע של OS/2 להשתלט על המסך. כאשר סביבת הביצוע של ה-DOS שולטת על המסך והמקלדת, היא נמצאת **בחזית** (foreground). כאשר המשתמש עובר מסביבת הביצוע של ה-DOS, היא עוברת **לרקע** (background).

כאשר סביבת הביצוע של DOS נמצאת בחזית, היא מתחלקת במעבד עם יישומים הנמצאים בסביבת הביצוע של ה-OS/2. כאשר מבוצע יישום DOS, המעבד נימצא במצב אמיתי (real mode). כאשר יישומי ה-OS/2 מבוצעים, המעבד נמצא במצב מוגן (protect mode). מערכת ההפעלה וה-Device Drivers שלה (תכניות השירות להפעלת יחידות קלט/פלט) נכתבו כך שיוכלו לטפל בעובדה שהמערכת יכולה לעבוד הן במצב אמיתי והן במצב מוגן. מצב זה ידוע כסביבה דו-מצבית (bimodal). למרות שמאפיין זה של ה-OS/2 הוסיף סיבוך למערכת ההפעלה ולפרמטרים שהיו צריכים להיכתב ל-OS/2, היה בו צורך כדי לתמוך בסביבת הביצוע של ה-DOS.

כאשר סביבת הביצוע של ה-DOS נמצאת ברקע, היא מושעית ולא מקבלת זמן מעבד. זו הסיבה מדוע אחת המגבלות על סביבת הביצוע של DOS היא תלות בתזמון וחוסר תמיכה ביישומים של זמן אמיתי. כאשר סביבת DOS נמצאת ברקע, יישומי OS/2 מתחלקים במעבד בינם לבין עצמם. ביצועי המערכת איטיים יותר כאשר סביבת הביצוע של DOS נמצאת בחזית, בגלל האופי הדו-מצבי של הסביבה.

המשתמש יכול לבחור את גודל סביבת ה-DOS בעזרת פרמטר בקובץ CONFIG.SYS. CONFIG.SYS הינו קובץ שה-OS/2 קוראת כאשר היא מותחלת. התכולה של קובץ זה קובעת רבים מהמאפיינים של ה-OS/2 ועל-פי קביעה של המשתמש. המשתמש יכול להגדיר ב-CONFIG.SYS שסביבת הביצוע של DOS לא תופעל.

חלק מה-OS/2, וכל ה-device drivers, נמצאים בזיכרון בכתובות שמתחת ל-640KB, והם מנצלים את הזיכרון שנועד לסביבת הביצוע של ה-DOS. מאחר והמעבד נמצא במצב אמיתי בזמן שיישום של DOS מופעל, יכול יישום של DOS לעקוף את הגנות המערכת של OS/2 ולבצע הוראות שאינן מותרות, או לכתוב על קודים השייכים למערכת ונמצאים מתחת לגבול של 640KB.

סביבת הביצוע של OS/2 מספקת מישק חדש לתכנות יישומים יחד עם רמה שלימה של תיפקודיות מעבר למה ש-DOS מאפשרת. המשתמש יכול לעבור מיישום OS/2 אחד למשנהו ולסביבת DOS בעזרת צירוף של מקשים "חמים". בעזרת בורר התכניות (Program Selector) אפשר לגשת ישירות ליישום או להתחיל יישום חדש.

כאשר יישום ה-OS/2 נמצא בחזית, הוא מתחלק במעבד עם כל יישומי ה-OS/2 הפעילים באותו רגע. המעבד נמצא במצב מוגן, ולכן, יישומים אינם יכולים

להכנס ללא הרשאה למרחב הכתובות של יישום אחר, למנוע מהמעבד לטפל בפסקים, או לפנות ישירות לאמצעי ק/פ (קלט/פלט). הוכנה תשתית לכך שיישומים מיוחדים יוכלו להכנס לרמה גבוהה מהרגיל של הרשאות, כדי שיוכלו להשיג הרשאת ק/פ (IOPL). נדון בכך בהמשך פרק זה, בסקירה על המעבד 80286. המשתמש חייב להגדיר יישומי IOPL ב-CONFIG.SYS כדי שיוכל להשתמש בהם. המנעות מטעינת יישומי IOPL בזמן נסיונות לאתר תקלה יכולה להיות דבר מועיל.

בהמשך ספר זה נעסוק באפשרויות החדשות ש-OS/2 מספקת ליישומי OS/2. טוב לדעת שרבים מיישומי DOS 3.3 טובים עדיין לשימוש בסביבת הביצוע של DOS בחסות OS/2. בצורה זו ניתן להנות עדיין מהתוכנה הישנה במשך התקופה שבה יתבצע מעבר הדרגתי ליישומים החדשים שמאפשרת סביבת הביצוע החזקה של OS/2.

מספר הגבלות ושיקולים ביישומי DOS

יישומי DOS אינם נתמכים בסביבת הביצוע של OS/2. אי אפשר להשתמש בפונקציות החדשות של OS/2 בסביבת הביצוע של DOS.

יישומי DOS התלויים בתזמון אינם נתמכים על-ידי סביבת הביצוע של DOS. אפילו כאשר סביבת ה-DOS נמצאת בחזית, היא **אינה** מקבלת 100 אחוז מזמן המעבד כפי שקורה בסביבת DOS 3.3 אמיתית. כך ייתכן, שיישום המבצע תיחקור (polling) לאמצעי קלט בסביבת DOS 3.3, לא יפעל בסביבת הביצוע של DOS תחת OS/2 בגלל בעיות בתזמון.

יישומים שמשתמשים בקריאה לפונקציות שאינן מתועדות על-ידי DOS 3.3 לא יקבלו תמיכה בסביבת הביצוע של DOS. יישומים הרגישים למספר הגירסה של DOS לא יפעלו מאחר ומספר הגירסה של סביבת הביצוע של DOS הוא 10.0. אין גם תמיכה ביישומים המשתמשים בקריאות לפונקציות תקשורת של DOS.

קיימות הגבלות בשימוש בפסקים (interrupts) מסוימים בתוכנות ה-DOS וה-BIOS בסביבת הביצוע של DOS. אין תמיכה ב-device driver הישן של DOS לטיפול בבלוקים. ישנם הגבלות משמעותיות על ה-device driver הישן ב-DOS לטיפול בהתקני התווים (character devices). ישנם גם הבדלים בין מספר תכניות שירות ופקודות ב-DOS 3.3 לבין תכניות השירות והפקודות הנתמכות על-ידי סביבת הביצוע של DOS ב-OS/2.

OS/2 אחראית לניהול משאבי המערכת. זו הדרך היחידה שבה יוכלו הרבה יישומים לנצל את אותה יחידה (כמו הדיסק) באופן יעיל. יש הגבלות על איזה פסקי חומרה יוכלו יישומי ה-DOS להשתלט ואיזה סוגי חומרה הם יוכלו לתכנת בעזרת גישה ישירה לחומרה.

יישומי DOS הנכנסים לתוך לולאה סתמית כדי להמתין שפסקים מסוימים יתרחשו בחומרה, אינם מאפשרים חלוקה יעילה של המעבד. יישומי DOS שיעשו את הדברים הבאים יגרמו לבעיה:

- יכתבו לתוך שטח בזיכרון שאינו בבעלותם.
- יכנסו לתוך לולאה סתמית ללא יכולת ביצוע.
- תכנות מחדש של בקר הפסקים (interrupts controller).
- שימוש או שינוי שעון ה-CMOS.

אם יישומים יבצעו משהו מהמתואר לעיל, או ינסו להשתמש במשאבי החומרה של המערכת ללא מעבר דרך מִמשק התכנות, הם מסתכנים בכך שלא ירוצו או יגרמו לכך שכל המערכת תפסיק לתפקד.

זיכרון אמיתי גדול וזיכרון בפועל

המעבד 8088, או המצב האמיתי במעבד 80286, תומכים באפשרות מיעון של זיכרון אמיתי עד 1MB. ה-8088 אינו תומך **בזיכרון בפועל** (virtual storage). **בזיכרון אמיתי** (real memory) אנו מתכוונים לזיכרון הפיסי במערכת ולכתובות הפיסיות שיש בזיכרון זה. **בזיכרון בפועל** (virtual storage) אנו מתכוונים לכתובות שבהן היישום משתמש ואותן הוא רואה. התפיסה של זיכרון בפועל אינה בשימוש כאשר ה-80286 נמצא במצב אמיתי. מרחב הכתובות בין 640KB ל-1MB מנוצל על-ידי מתאמים לקלט/פלט ממופה-זיכרון וה-ROM BIOS. לכן המצב האמיתי של ה-80286 יכול לתמוך בקודים של יישומים ובנתונים עד ל-640KB. זאת אומרת, שיישומי ה-DOS אינם יכולים לפגוע במערכת ההפעלה הנמצאת בכתובות מעל 640KB.

במערכת הפועלת ב-DOS, אחת הדרכים להשתמש בזיכרון מעל 1MB היא להשתמש בדיסק בפועל (virtual disk - VDISK). הדיסק בפועל מוגדר בזיכרון והוא מהיר בצורה משמעותית לעומת הדיסק הקשיח. יישום יכול להשתמש בו בדיוק כמו שהוא משתמש בדיסק רגיל (דרך קובץ מערכת בשם API). היישום יכול לעבד ישירות כל נתון הנמצא בכתובות שמתחת ל-640KB. לשימוש בזיכרון שמעבר ל-1MB יש הרבה צדדים שליליים. היישום חייב לנהל את הנתונים שלו בדיסק בפועל בצורה של קובץ, בניגוד לנתונים הנמצאים בזיכרון. הוא חייב לאחסן נתונים ולהביא נתונים חדשים בכל פעם שהוא צריך להשתמש בנתונים הנמצאים על הדיסק בפועל במקום בזיכרון. פקודות היישום הנמצאות על הדיסק בפועל אינן יכולות להתבצע בצורה ישירה. הן חייבות להיות מובאות לזיכרון כמו שמובא רובד (overlay) ואז להתבצע. הדבר יכול לגרום לעיכובים שלא ניתן לקבל אותם במערכות זמן אמת ולהוסיף מורכבות משמעותית למבנה ולתכנון של היישום.

ניתן להשתמש בבנק זיכרון כדי לנסות להתגבר על צוואר הבקבוק של 640KB. שימוש זה ביותר מ-640KB טוב יותר מכלום, אבל הוא אינו הרחבה טבעית לאפשרויות הזיכרון של ה-80286 והוא לא מאפשר ליישומים להתרחב בצורה טבעית לתוך ארכיטקטורת האפשרויות של 80286 ו-80386. זו אחת הדרכים שבהן מערכות 8088 מתמודדות עם הצורך לאפשר ליישומים להשתמש ביותר מ-640KB של זיכרון. למערכות המבוססות על המעבד 8088 אין אלטרנטיבות אחרות.

OS/2 מספקת סביבה שבה אפשרויות המיעון של הזיכרון בעזרת המצב המוגן של 80286 יכולות להיות מנוצלות, לא רק על-ידי מערכת ההפעלה, אלא גם על-ידי היישומים. בסביבה זו, ל-OS/2 וליישומי ה-OS/2 יש גישה ישירה ל-16MB של זיכרון פיסי בלי עזרה של אמצעים לא טבעיים כמו דיסק בפועל, או בנק זיכרון.

בנוסף לכך, OS/2 משתמש באפשרויות של זיכרון בפועל בסביבת המצב המוגן של ה-80286. במקרה זה, הכתובות שהיישום רואה אינן שוות לכתובות הפיסיות. הדבר מאפשר לכל היישומים שנמצאים באותו רגע בזיכרון להשתמש בהרבה יותר זיכרון מאשר קיים באמת במערכת הפיסית. ליישום נדמה שכל הזיכרון קיים באמת, למרות שהוא לא קיים במציאות. OS/2 מנצלת את העובדה

שחלק מהזיכרון הזה אינו דמיוני ונמצא באמת על הדיסק, והיא תביא את הזיכרון הזה לתוך הזיכרון הפיסי האמיתי כאשר היישום ינסה להכנס אליו. ליישום אין שום ידיעה על-כך שהפקודות, או הנתונים, לא היו באמת בזיכרון הפיסי כאשר הוא ניסה לבצע אותן או להכנס אליהן. תהליך זה נקרא תמיכה בזיכרון בפועל.

נדון בתפישות עבודה אלו ביתר פירוט בהמשך פרק זה, כאשר נסקור את ארכיטקטורת ה-80286, ובפרק על אפשרויות הטיפול בזיכרון (פרק 2).

כבר הדגשנו שחשוב שיהיה ליישומים זיכרון גדול ככל האפשר. עבור OS/2 יש לכך חשיבות רבה יותר כי היא תומכת במספר יישומים בו-זמנית, ועליה לספק מרחבי כתובות גדולים לכל יישום. אפשרויות הזיכרון בפועל ב-80286 מאפשרות ל-OS/2 לעשות זאת. על-ידי תמיכה באפשרויות הניצול של זיכרון אמיתי גדול שמאפשרת ארכיטקטורת ה-80286, OS/2 מאפשרת שימוש ישיר ואופטימלי בזיכרון אמיתי גדול הקיים במערכות מחשב אישיות כיום ושיהיה קיים בעתיד.

תמיכה בזיכרון-יתר

כפי שראינו, OS/2 מאפשרת ליישום או ליישומים להיות גדולים יותר מבחינת השימוש בזיכרון, מאשר הזיכרון האמיתי במערכת. מאפיין זה נקרא תמיכה **בזיכרון-יתר** (memory overcommit). OS/2 משיגה זאת על-ידי **העברת סגמנטים** (swapping segments) מהזיכרון לדיסק כאשר לא משתמשים בהם (אנו נדון בסגמנטים בסקירה על הארכטיקטורה של 80286 בפרק זה). הסגמנטים שהשתמשו בהם הכי פחות לאחרונה (**least recently used**) יועברו ראשונים לדיסק. לאחר שסגמנט הועבר לדיסק (swapped out), הזיכרון האמיתי שהיה תפוס על-ידו יתפנה לשימוש חוזר ואפשר לאחסן בו סגמנט אחר שיובא מהדיסק (swapped in). הסגמנט יועבר פנימה מכיון שמערכת ההפעלה, או היישום, החליטו לבצע אותו או לגשת אל הנתונים השמורים בו.

כדי להפטר מהסגמנטים בזיכרון שהשתמשו בהם הכי פחות לאחרונה, אפשר לא להעביר סגמנטים החוצה, אלא **לבטל** (discard) אותם במצבים הבאים:

- היישום יצר את סגמנט הנתונים אם אפשרות לביטולו.
- הסגמנט אינו יכול להשתנות מכיון שהוא סגמנט המכיל הוראות.

אם צריך להשתמש בסגמנט שימוש חוזר אז אפשר:

- להביא אותו חזרה מהמקור.
- ליצור אותו מחדש על-ידי היישום. היישום נתן למערכת רשות להפטר מסגמנט הנתונים.

גודלם של הסגמנטים יכול להשתנות עד גבול של 64KB, ולכן השטח שהשתחרר כתוצאה מביטול והעברה של סגמנטים מתוסף למרחב הכתובות האמיתי. הבה נניח שיש בזיכרון האמיתי שני שטחים בגודל של 32KB וישנו סגמנט בן 64KB שצריך להעבירו פנימה. במקרה כזה המערכת לא תוכל להעביר את הסגמנט לתוך שטחים שאינם רציפים בזיכרון האמיתי. היא תחליט להזיז סגמנטים בזיכרון האמיתי במטרה לצרף מספר שטחים זה לזה וליצור על-ידי כך שטח פנוי גדול יותר. תהליך זה נקרא **הזזת סגמנטים** (segment motion).

אפשרויות ניצול הזיכרון באמצעות ה-80286 במצבו המוגן ·מאפשר ליישומים גדולים להתקיים בו מבלי להיות מודעים לגודל הזיכרון העומד לרשותם. מצב זה שונה לחלוטין מהמבנים השכבתיים שיישומים גדולים היו צריכים לטפל בהם בעבר, כאשר היו גדולים יותר מהזיכרון שעמד לרשותם. היישומים היו צריכים להיות מתוכננים בגודל מסוים ובנויים בדרך מסוימת כדי שיוכלו להביא בעת הצורך, את ההוראות שהיו בתור לביצוע, אל מרחב הכתובות שלהם. יישומים שהשתמשו בשכבות היו צריכים לקחת בחשבון שמרחב כתובות המוקצה לנתונים בא על חשבון מרחב כתובות להוראות התכנית ולהיפך. בסביבה של תמיכה בזיכרון-יתר וזיכרון בפועל, יישומים אינם צריכים לעסוק בארגון ובמבנה אלא רק בפרטים הבאים:

- במספר הסגמנטים שהמערכת ממליצה שיהיו להם,
- מספר הסגמנטים המקסימלי שבו המערכת יכולה לתמוך,
- ההשלכות על הביצוע כתוצאה מכמות ההעברות שהמערכת תבצע.

סביבה רבת יישומים

ישנם שיפורים בסביבת DOS המאפשרים תמיכה בסביבה רבת יישומים. אולם, שיפורים אלה אינם עולים בקנה אחד עם רבות מההגבלות הקיימות במצב האמיתי ב-80286. קשה לתמוך בצורה יעילה ביישומים רבים של סביבת DOS משופרת בזיכרון של 640KB. אפשר להשתמש בתחבולות כדי להעביר יישומים לתוך דיסק בפועל וממנו, אבל הביצוע הסופי יכול להוות בעיה. בנוסף לכך, המצב האמיתי ב-80286 אינו מגן על היישומים זה מפני זה, ואין הוא מגן על מערכת ההפעלה מפני היישומים.

בעיות אלו נעלמות בסביבת OS/2. המערכת מספקת מרחב כתובות דמיוני גדול לכל יישום המופעל במסגרתה. OS/2 משתמשת במנגנוני הגנה של ה-80286 כדי לספק הגנה בין יישומים לבין מערכת ההפעלה ליישומים.

כמו כן, OS/2 מספקת ניהול משאבים וכללים הכרחיים כך שיישומים מתוחכמים רבים יוכלו לפעול זה בצד זה באותה מערכת ללא תקלות. אפשר לתת ליישומים עדיפות ביצוע, כדי שיקבלו את הכמות הדרושה של זמן המעבד.

OS/2 מאפשרת ליישומים רבים לפעול במקביל. זאת אומרת שמשתמשים יכולים להגדיל בצורה משמעותית את פריון העבודה שלהם אם הם יתחילו להשתמש במחשבים שלהם בצורה יותר נבונה: אפשר לקבל נתונים ממקור מרוחק בעזרת תוכנת תקשורת ובאותו זמן לבצע חישובים בגיליון אלקטרוני גדול, בעת שהמשתמש מפעיל מעבד תמלילים בחזית ומדפיס תזכיר. המשתמש יכול לעבור מיישום אחד למשנהו כדי לעקוב אחר הפעילות בכל אחד מהם.

OS/2 מאפשרת שימוש במגוון רב יותר של אפשרויות בתוכנה, כי היא מספקת סביבה שבה יישומים יכולים לפעול זה בצד זה ולהתבצע מבלי להפריע זה לזה. קודם לכן, המשתמש אולץ לרכוש מערכת משולבת של יישומים שיכלו להתבצע יחד בצורה נכונה בסביבת ה-DOS. בעולם החדש של OS/2, כללי פיתוח היישום מאפשרים ליישומים שפותחו על-ידי משווקים שונים לפעול יחדיו. הדבר פותח למשתמש אפשרות לבחור את היישום הטוב ביותר לכל בעיה.

סביבה רבת משימות

ביצוע רב-משימתי הוא מצב, שבו המערכת מספקת ליישום סביבה, שבה הוא נראה למשתמש כעושה יותר מפעולה אחת באותו זמן. במערכת יש רק מעבד ראשי אחד, ולכן אין היא יכולה לבצע בפועל יותר מאשר משימה אחת בו-זמנית. אולם, הרבה פעמים קורה שביישום יש חלקים הניתנים לביצוע במקביל, בתנאי שקיימות נקודות סינכרון המוגדרות היטב. קורה גם שהמעבד אינו יכול להמשיך לבצע חלק של עבודה מסוימת, כי עליו לחכות לביצוע קלט/פלט מהתקן חיצוני איטי, כמו מקלדת או דיסק. סביבה רבת משימות מאפשרת לחלקי העבודה להיות בנויים בצורה כזאת, שבזמן שחלק אחד של העבודה ממתין שמשהו יתרחש במערכת, חלק אחר יורשה להשתמש במעבד. בסביבה רבת יישומים, יש בכל יישום חלקים של עבודה המחלקים ביניהם את זמן המעבד.

DOS מתוכננת להיות סביבה חד-משימתית, אשר בה המערכת יכולה לבצע רק חלק אחד של העבודה בו-זמנית ורק ביישום אחד. כאשר יישום מצפה לנתונים מהדיסק, הוא ממתין להם ואינו עושה דבר אחר בזמן ההמתנה.

יכול להיות שתתהה על כך שתכניות מסוימות בסביבת ה-DOS נראות לך רב-משימתיות. נכון שעל-ידי עקיפת המישקים הרגילים של DOS, אפשר לבנות תכנית שיהיו לה מאפיינים של ביצוע רב-משימתי בסביבת ה-DOS. אולם, סביר להניח שיישום שיעשה זאת, לא יפעל עם יישומים אחרים, במיוחד אם יישומים אלה יעקפו גם הם את מישקי DOS הרגילים, כדי לאפשר לעצמם מאפיינים של ביצוע רב-משימתי.

הביצוע הרב-משימתי ב-OS/2 מאפשר שלכל יישום בסביבה רבת יישומים יהיו מאפיינים של ביצוע רב-משימתי. כל יישומי OS/2 יפעלו זה בצד זה ויתבצעו בו-זמנית ללא הפרעה זה לזה. כמובן שככל שיהיו יותר יישומים, כך יקבל כל יישום פחות מזמן המעבד. ב-OS/2 ניתן לתת ליישומים עדיפויות, כדי שיישומים שבעיית הזמן אצלם היא קריטית, יקבלו זמן מעבד בעדיפות לעומת יישומים רגילים.

יחידת העבודה הבסיסית הניתנת לביצוע ב-OS/2 נקראת **thread**. thread מורכב מסידרה של הוראות תכנית שיתבצעו במערכת. ב-thread יש מאפיינים הדומים לתכנית DOS פשוטה. יחידת הבעלות הבסיסית על משאבי המערכת נקראת **תהליך (process)**. כל תהליך מורכב מ-thread אחד או יותר. מספר threads של אותו תהליך מתחלקים במערכת משותפת של משאבי מערכת (פרט לדברים כמו מחסנית, stack). נדון במאפיינים אלה בפירוט בפרק על ביצוע רב-משימתי וסביבה רבת יישומים (פרק 3).

ביצוע רב-משימתי נחשב לצורך חיוני בסביבת המערכות החדשה של OS/2. ללא מאפיין זה, לא היה אפשר לפתח יישומים מתוחכמים שהיו יכולים לפעול זה בצד זה בסביבה רבת יישומים. משתמשים מצפים ליישומים שיוכלו להגיב במהירות לקלט מהמקלדת כאשר בו בזמן הם ימשיכו במסלולי עיבוד אחרים בתכנית. ככל שהמעבדים הופכים להיות מהירים יותר, המהירות היחסית שלהם הופכת לגבוהה יותר ביחס לאמצעים חיצוניים שבהם הם תומכים. תת-מערכות תקשורת ויישומים בזמן אמיתי דורשים שבמסגרת של ביצוע רב-משימתי יהיה ניתן להשגיח על אירועים חיצוניים בזמן שמבצעים חלקי תכנית אחרים בעדיפות נמוכה יותר. ללא ביצוע רב-משימתי, לא היה ניתן לבצע דבר ממה שתואר לעיל בסביבה רבת יישומים.

תקשורת בין תהליכים

thread הינה יחידת הביצוע הבסיסית בסביבת OS/2. מספר threads יוצרים תהליך. בסביבת הביצוע הרב-משימתי ב-OS/2, אם כל ה-threads קשורים לאותו תהליך, קל להם לפעול בצורה מתואמת ולהסתנכרן זה לזה בכל זמן. ה-threads מתחלקים באותם משאבי מערכת.

באופן טבעי, תהליכים שונים אינם מתחלקים באותם משאבי מערכת. אי לכך קשה לתהליכים שונים להעביר ביניהם מידע, או להסתנכרן בצורה מתאימה (אם זה נדרש). מצב זה מוביל לצורך להשתמש בתקשורת בין תהליכים.

OS/2 מספק הרבה מנגנונים המאפשרים לתהליכים שונים להתקשר ביניהם ולהסתנכרן זה לזה:

- אתת RAM (semaphore) ואתת מערכת.
- שיגרה לטיפול באותות (signal handler).
- צינורות (pipes).
- תורים (queues).
- זיכרון משותף (shared memory).

נדון בתפישות אלו בהרחבה בפרק על ביצוע רב-משימתי וסביבה רבת יישומים (פרק 3). על-ידי שימוש במספר רב של אפשרויות לתקשורת בין תהליכים. OS/2 מספקת סביבה שבה יישומים כמו גם תת-מערכות יכולים להיות מתוכננים בהתאם לתפקיד שאותו הם צריכים לבצע, ולמאפיינים שהם צריכים להציג. אם יחידות העבודה השונות הם תהליכים, אז הם מטבעם מוגנים זה מזה ומשפיעים זה על זה רק דרך קווי תקשורת מוגדרים היטב.

מישק קישור דינמי ותמיכה בשפות עיליות

ב-OS/2 יש תמיכה ב-6 שפות תכנות המשווקות על-ידי יבמ:

- קובול/2
- פורטרן/2
- C/2
- מקרו אסמבלר/2
- פסקל/2
- בייסיק/2

שפות אלו תפעלנה הן בסביבת DOS 3.3 והן בסביבת OS/2, ולכן ניתן לכתוב תכניות שתפעלנה בשתי סביבות הביצוע. לצורך כך יש להשתמש בפונקציות מיוחדות של OS/2 ויש לדאוג לכך שהפקודות תהיינה תואמות לכללים מסוימים. תת-קבוצה זו של תוכנת OS/2 API נקראת **מישק לתכנות יישומי משפחה** (Family Application Programming Interface). השימוש במישק זה מאפשר להעלות את פריון העבודה בעת פיתוח יישומים שיוכלו לרוץ הן ב-DOS 3.3 והן ב-OS/2 (כיישום OS/2 במצב המוגן), בתנאי שאין צורך במאפיינים החדשים של OS/2, ובתנאי שהיישומים יוכלו להשתמש במישק זה.

שלוש השפות הבאות (מתוך הרשימה הקודמת) מאפשרות להשתתף **במישק התכנות האחיד** (Common Programming Interface), שהוא אחד המרכיבים **בארכיטקטורת היישומים הבין-מערכתית** (Systems Applications Architecture - SAA) של

יבמ:

- קובול/2
- פורטרן/2
- C/2

אם אתה מתעניין בסביבת המיחשוב ואפשרויות השימוש במישק שמספקת ארכיטקטורת המערכות של יבמ, אז השיקול של בחירת השפה המתאימה ליישום הוא שיקול חשוב.

המישק לתכנות יישומים (Application Programming Interface - API), משתלב ביתר טבעיות עם מבנים האופייניים לתכנות בשפות גבוהות. אופי הקישור הדינמי של מישק זה מוסיף מימד חדש של גמישות למישק של מערכת ההפעלה, בהיותו כלי רב עוצמה. יישומים מתייחסים לפקודות חיצוניות כקריאות רחוקות (far calls) ולכן אינם יכולים להכנס למעשה לפקודות אלה, אלא עד לאחר שהתכנית תוטען לזיכרון, או עד אשר היא תפעל.

פקודות מערכת ההפעלה התומכות במישק קישור דינמי מסוים ניתנות לשינוי תוך שמירת תאימות עם יישומים קיימים, ללא צורך לבצע מחדש הידור או קישור. בנוסף לכך, מישק הקישור הדינמי ניתן להרחבה לשירותים שתת-מערכות או יישומים אחרים ירצו לספק למערכת הכללית. זו הסיבה מדוע הקישור הדינמי שמאפשר המישק לתכנות יישומים ב-OS/2 הוא מודולרי וניתן להרחבה. אפשר להוסיף קבוצות שלמות של תכניות שירות למערכת בצורה שתראה כאילו הם חלק בלתי נפרד של המערכת, אפילו אם הן מסופקות על-ידי משווקים עצמאיים, או על-ידי קבוצה שונה בתוך הארגון עצמו. דוגמה טובה לכך יכולה להיות תת-מערכת התקשורת, אשר יכולה להיות מורכבת בצורה כזו, שיישומים יתייחסו אליה כאל הרחבה טבעית של מערכת ההפעלה.

המאפיינים של מישק הקישור הדינמי מאפשרים דברים נוספים. בסביבת DOS הישנה, כאשר תכנית קושרה (linked) עם קבוצה של פקודות באמצעות הפניות חיצוניות, קטעי התכנית החיצוניים הפכו לחלק מהתכנית מאותו רגע והלאה. אם השגרות החיצוניות שונו לאחר מכן, היה צורך לקשר את היישום מחדש. השגרות החיצוניות האלו הופכות לחלק בלתי נפרד של היישום לאחר הקישור, ונגררות איתו תמיד.

קישור דינמי משנה את כל התמונה. לאחר שהקישור בוצע, היישום אינו קשור לשגרות החיצוניות, והן אינן מהוות חלק ממנו. רק כאשר היישום מופעל במערכת, פונים ומשתמשים בשגרות הקשורות אליו. למאפיין זה יש יתרונות רבים ההופכים את סביבת הקישור הדינמי ב-OS/2 לטובה יותר מכל דבר אחר שקדם לה.

אפשר לשפר את התיפקוד של מישק הקישור הדינמי תוך שמירת התאימות עם יישומים ישנים שעברו כבר את תהליך הקישור. היישומים החדשים אינם צריכים להכתב מחדש או להתקשר מחדש. הפקודות העומדות מאחורי מישק הקישור הדינמי ניתנות לשינוי כללי, כדי לאפשר תמיכה בסביבה של מערכות שונות. גם אז היישום אינו צריך להתקשר מחדש כדי לפעול באותה סביבה חדשה. הפריט היחיד שחייב להישאר ללא שינוי הוא המידע שמרכיב את ההפנייה החיצונית לספריית הקישור הדינמי ולשיגרה בתוך הספריה. מידע זה אינו דרוש ליישום עד למועד שבו הוא מוטען לצורך ביצוע, או עד לרגע שבו הוא מבוצע למעשה.

קישור דינמי מונע את הפיכת השגרות החיצוניות לחלק בלתי נפרד מהיישום.

במקרה של הפניות חיצוניות למישק הקישור הדינמי, תכנית היישום אינה מכילה למעשה את השגרות שאליהן היא מחוברת בצורה דינמית. למעשה, היא מכילה רק את ההפניות אליהן. המערכת תפענח קשרים אלה בזמן הטעינה או ההרצה.

שגרות הקישור החיצוניות אינן הופכות לחלק מהיישום עד הרגע שבו משתמשים בו. לכן רק מערך אחד של שגרות צריך להיות דרך קבע בזיכרון גם אם יישומים רבים מתחברים לאותו מישק קישור דינמי. בכך משיגים חיסכון רב בשטח האחסון החיצוני הדרוש לאותם יישומים.

יתרונות הקישור הדינמי אינם מסתיימים כאן. אם הפקודות התומכות במישק מסוים לקישור דינמי נמצאות כבר בזיכרון האמיתי כתוצאה משימוש ביישום קודם, אז כאשר יישום חדש מנסה להשתמש במישק, המערכת תבין שהפקודות התומכות במישק זה נמצאות כבר בזיכרון והיא תשתמש בפקודות שבזיכרון, במקום לפנות לדיסק ולהביא אותן שוב. זה משפר את ביצועי המערכת ואת מאפייני השימוש בזיכרון בסביבה רבת יישומים.

יישומים יכולים להשתמש בקישור דינמי בשתי דרכים שונות: **בעת טעינה** (load time) או **בעת ריצה** (run time). השימוש בזמן טעינה הוא מה שרוב האנשים מכירים. במקרה זה כל ההפניות המקושרות בצורה דינמית (dynamic link) מוחלפות בזמן שהיישום מוטען לראשונה לתוך המערכת כדי להתבצע. בזמן זה, הפקודות של המישק המקושר בצורה דינמית יוטענו או לא יוטענו לזיכרון בהתאם למאפיינים של הסגמנטים שלהם (**הטען מראש, או הטען לפי דרישה**, שידונו בפרק 2) ואם הכמות הפנויה של הזיכרון הפנימי מספיקה עבור כל הסגמנטים שיש צורך להטעינם. נדגיש שוב שיש צורך להמיר את ההפניות החיצוניות כאשר התכנית מוטענת לראשונה.

קישור דינמי בזמן ריצה מוסיף מימד חדש לגמישות של הקישור הדינמי. כאשר משתמשים בקישור דינמי בזמן ריצה, היישום אינו מחליט במפורש באיזה מישק הוא רוצה להשתמש, עד שפקודות היישום מבוצעות למעשה. קיימים מישקי מערכת המאפשרים קישור דינמי לשיגרה מסוימת בספריה במשך זמן הביצוע של התכנית. לדוגמה, תכנית יכולה להיכתב כך שתשאל את המשתמש באיזה חבילת קישור דינמי צריך להשתמש כדי לבצע פעולה מסוימת. המשתמש יוכל להזין את המידע הדרוש תוך כדי ביצוע התכנית, אשר תוכל להשתמש במידע הזה כדי להתחבר למישק חיצוני ולהעביר לו את השליטה על המשך הביצוע.

תאימות עם מערכת הקבצים של DOS

מערכת ניהול הקבצים ב-OS/2 משתמשת בטבלת הקצאת הקבצים (File Allocation Table - FAT) בדיוק כמו ה-DOS. ל-OS/2 יש תמיכה דומה במחיצות של הדיסק הקשיח, אשר יש צורך בהן בדיסקים קשיחים בעלי נפח אחסון הגדול מ-32MB. תאימות מערכת הקבצים עם DOS הינו מאפיין חשוב במיוחד של OS/2.

תודות לתאימות זו, ניתן להעביר נתונים מ-DOS ל-OS/2 ללא צורך בשינויים. אם המשתמש מריץ יישום DOS המפיק נתונים לצורך אחסון בדיסק קשיח או דיסקט, אז הנתונים האלה ניתנים לשימוש מיידי כקובץ OS/2. כללי מתן השמות לקבצים בשתי המערכות זהים, והם מאפשרים למשתמש לשמור על מערכת עיקבית של שמות בשתי מערכות ההפעלה.

ניהול מבוקר פסקים של התקנים חיצוניים

אמצעים חיצוניים נתמכים בסביבת ה-OS/2 על-ידי תכניות שירות הנקראות **device drivers**. בסביבת ה-DOS אפשר לתמוך באמצעים חיצוניים בצורה של **חקירה** (polling) מכיון ש-DOS הינה סביבה חד-יישומית וחד-משימתית. בנוהל חקירה, כאשר יישום או device driver מנסה להגיע אל אמצעי מסוים לצורך ביצוע פעולה כלשהי ונכשל, הוא ימשיך לחקור את האמצעי החיצוני בעזרת לולאה אין-סופית, עד אשר האמצעי החיצוני יסיים את הפעולה ויתפנה. כל פעולה אחרת לא תתבצע במערכת עד לתום החקירה. התכנית נמצאת בזיכרון ושואלת כל הזמן "האם גמרת?" חקירה מבזבזת כמות גדולה מאוד של משאבי עיבוד והיא אינה מתאימה לסביבה רבת יישומים ורבת משימות, שבה חשוב להשתמש בכל משאבי המערכת בצורה יעילה ככל האפשר.

הגירסה של ה-device drivers ל-OS/2 נבנתה כך שתוכל להשתלב בסביבה רב-משימתית. ה-device driver ב-OS/2 אינו מחכה להתקן החיצוני שיגמור לבצע את מה שנתבקש לעשות. מה שקורה בפועל הוא, שהמערכת מבצעת עבודה אחרת, והתכנית מחכה שיהיה **פסק** (interrupt) בהתקן החיצוני. פסקים הם הדרך שבה התקנים חיצוניים עוצרים את המערכת ומאלצים אותה לשנות את הפקודות שאותן היא מבצעת באותו רגע. המערכת יכולה לבצע פקודות של יישומים עד אשר ההתקן החיצוני מבקש שישרתו אותו על-ידי הפקה של פסק. נדון שוב בפסקים בסעיף בפרק זה העוסק בארכיטקטורה של ה-80286 ובפרק על ניהול מבוקר פסקים של התקנים חיצוניים (פרק 6).

DOS אינה מספקת מנגנון כללי המאפשר לפקודות של תכנית יישום להתבצע בזמן שהמערכת ממתינה להתקן חיצוני שישלים את עבודתו. העובדה ש-OS/2 מסוגלת לעשות זאת היא הסיבה העיקרית ליכולת שלה להשתמש במערכת בצורה אופטימלית. מספר התקנים חיצוניים יכולים לבצע עבודה בו-זמנית עבור אותו יישום, או עבור יישומים שונים. בזמן ש-device driver מסוים מחזיר את השליטה למערכת ומחכה לאירוע שיקרה בהתקן החיצוני, תכנית היישום יכולה לבקש שירות מהתקן חיצוני אחר. ההתקן האחר יכול להתחיל את עבודתו לפני שהראשון משלים את פעולתו. ניתן לגרום לכך שהדיסקט, הדיסק הקשיח, המדפסת, כניסות התקשורת וכו' יוכלו לעבוד בו-זמנית. מאפיין זה מגביר את העוצמה של המערכת, הרבה מעבר לאפשרויות של DOS.

תת-מערכות לקלט/פלט (תצוגה, מקלדת ועכבר)

OS/2 מספקת תמיכה באמצעי קלט/פלט כמו תצוגה, מקלדת ועכבר באמצעות שילוב של תכניות להגדרת אמצעי קלט/פלט ומישקי קישור דינמיים. תת-מערכות אלו משולבות לתוך מאפייני מישק המשתמש של המערכת, ומספקות מישקים מיוחדים שמקלים על השימוש באמצעי הקלט/פלט החיצוניים.

תת-מערכת התצוגה מספקת מערך של מישקים ברמה גבוהה יותר ממישקי התצוגה BIOS Int 10h שקיימים בסביבת DOS. תת-מערכת המקלדת מאפשרת ליישום לקבל כקלט הן scan code והן מייצגים של קוד ASCII. הדבר מתוכנן כך שהיישומים לא יזדקקו ל-CPU כאשר הם מחכים להקשה מהמשתמש. תת-מערכת העכבר מאפשרת ליישום לנהל בצורה כללית את העכבר. היא יכולה לשלוט בדרך שבה יטופל הסמן של העכבר, ואיזה אירועים המופקים על-ידי העכבר יכללו כנתונים באוגר הקלט שלו.

כל אחת מתת-מערכות אלו מספקת פונקציות המאפשרות ליישום, או לתת-מערכת

אחרת, לייַרט פונקציות קיימות. על-ידי השתלטות על הפונקציות של התצוגה, המקלדת והעכבר, יישום יכול להרחיב או להחליף את הפונקציות הבסיסיות המסופקות על-ידי תת-מערכות אלו.

תמיכה בשפות לאומיות ואמצעים לטיפול בהודעות

אנו חיים בתקופה שבה חברות שיווק מצפות שתהיה להן גמישות באספקת תוכנה לשם שימוש במדינות שונות, כפי שמכתיבים הצרכים העסקיים שלהן. משווקי תוכנה עצמאיים רוצים שהמוצרים שלהם יהיו משווקים במדינות רבות תוך השקעה מינימלית בשינויים כתוצאה משפה, או ממידע האופייני למדינה זו או אחרת. הדבר יאפשר להם להחזיר חלק גדול יותר של ההשקעה בתוכנה המקורית.

OS/2 תומך ב-11 שפות לאומיות עם המקשים הקשורים אליהן. זאת אומרת שכל מסכי ה-OS/2, ההודעות והתיעוד מתורגמים לשפות המתאימות ושהתמיכה במקלדת מכירה את מערך (סידור) המקשים המתאים. השפות הן:

- דנית.
- הולנדית.
- צרפתית.
- גרמנית.
- איטלקית.
- נורווגית.
- פורטוגזית.
- ספרדית.
- שבדית.
- אנגלית בריטית.
- אנגלית אמריקאית.

OS/2 תומכת גם בשישה מערכי מקלדות נוספים:

- בלגית.
- קנדית.
- פינית.
- ספרדית דרום-אמריקאית.
- צרפתית שוויצרית.
- גרמנית שוויצרית.

התמיכה בעברית זמינה על-ידי יבמ ישראל.

OS/2 תומכת גם בשינוי **דף קוד** (code page) לצורך צגים ומדפסות מיוחדים. דף קוד יכול להחשב כמערכת התווים, או תבנית התווים (fonts) של הצג או המדפסת. תמיכה זו מאפשרת ליישומים לבחור את מערכת התווים לצג ולמדפסת המתאימה ליישום מסוים. דף קוד המתאים ליישום מסוים לא חייב להיות זהה לשפה שנקבעה בקובץ CONFIG.SYS. כדי לטפל בזה, OS/2 מספקת תמיכה לשני דפי קוד בו-זמנית, אשר ניתנים לבחירה בזמן תהליך ההתקנה. המשתמש והיישום יכולים לבחור בדף הקוד המתאים כשיש צורך בכך. דפי הקוד הנתמכים ב-OS/2:

- 477 - PC ASCII, ארה"ב ומדינות אחרות.
- 850 - ריבוי שפות (multilingual), מדינות אירופאיות רבות.
- 860 - פורטוגזית.
- 863 - צרפתית קנדית.
- 865 - שפות נורדיות.

פרט למעבר מדף קוד אחד למשנהו, קיימים מאפיינים נוספים לתמיכה בשפות לאומיות המסופקים על-ידי OS/2. מאפיינים אלה הופכים את כתיבת התכניות בסביבה בינלאומית לקלה יותר. הם באים לידי ביטוי בעת הטיפול בהודעות של OS/2. הם מסוגלים להפריד בצורה מוחלטת את ההודעות שהיישום משתמש בהן בתכנית לבין ההודעות המוצגות למשתמש. ההודעות נמצאות בקובץ נפרד הניתן לתרגום בצורה עצמאית על-ידי אנשים ללא ידע במיחשוב, אך בעלי ידע בשפות זרות. זו דרך מאוד יעילה לפתח תוכנה למדינות שונות מבלי לבצע הידור מחדש, או קישור מחדש לתוכנה. המתרגמים עצמם יכולים ליצור קובץ הודעות חדש בעזרת כלים פשוטים לעריכת קבצים.

גם מעצבי תוכנה יכולים להשתמש בטכניקת התרגום כדי להפריד את השפה שתוצג למשתמש הסופי ממה שהמתכנתים עובדים עליו. המתכנתים יוכלו להמשיך את מה שהם עושים בעוד שמומחים בהנדסת אנוש יוכלו להתרכז בניסוח הטוב ביותר של ההודעות למשתמש. חלוקת העבודה בדרך זו תומכת בפיתוח תוכנה ידידותית למשתמש.

הגירה של תכניות DOS ל-OS/2

תכנית יישום אשר פועלת תחת DOS לא תפעל בסביבת הביצוע של OS/2. הדרך שבה תכנית מתחברת למערכת שונה בשתי מערכות ההפעלה. ב-DOS, תכניות משתמשות במישק ה-INT, כאשר ב-OS/2 תכניות משתמשות במישק הקישור הדינמי.

כדי שנוכל להריץ יישום DOS שנכתב בשפה עילית כיישום OS/2, חייבים לבצע הידור וקישור מחדש של התכנית. סביר להניח, שיהיה גם צורך לבצע שינויים בתכניות מקור, ובוודאי שיהיה צורך לבצע שינויים בתכניות הכתובות בשפת אסמבלר. יהיה צורך לשנות את כל קריאות ה-INT לקריאות קישור דינמיות המופנות למישק לתכנות יישומים ב-OS/2. התיעוד הטכני הקיים מספק הנחייה איך לעבור משגרות הביצוע של DOS INT למישק לתכנות יישומים ב-OS/2.

תהליך ההסבה של תכנית DOS לתכנית היכולה לפעול בסביבת הביצוע של OS/2 הינו תהליך הגיוני, אפילו אם התכנית אינה משתמשת באף אחת מהאפשרויות החדשות ב-OS/2. זאת, מאחר שתכנית ב-OS/2 יכולה להתקיים בסביבה רבת היישומים של OS/2, וניתן להשתמש בה מבלי להתייחס למגבלות זיכרון.

אפשר לשפר את רוב היישומים על-ידי ניצול של פונקציות חדשות הקיימות ב-OS/2. לדוגמה, תכניות גיליון אלקטרוני ניתנות לשיפור בכך שהן יתמכו בנפחים גדולים של נתונים מבלי להעביר נתונים אל הדיסק וממנו. הגיליון האלקטרוני ייראה ליישום כאילו כולו נמצא בזיכרון. במציאות, אם למערכת אין מספיק זיכרון פיסי, OS/2 תעביר חלק מהגיליון אל הדיסק. המשתמש יוכל לרכוש זיכרון פיסי יותר גדול ועל-ידי כך להמנע מכל העברה של נתוני הגיליון. תכנית הגיליון האלקטרוני עצמה לא שונתה, אך מאפייני הביצוע שלה שונו לטובה בצורה משמעותית.

משווקי תוכנה עצמאיים יוכלו לשפר את היעילות ואת יכולת התחרות של התוכנה שלהם על-ידי ניצול רמה חדשה של פונקציונליות הקיימת בסביבת הביצוע של OS/2. ניתן לצפות שבמשך הזמן, בעקבות הלחץ שמפעילים משווקים ובעקבות הלחץ שמפעילים משתמשים הרוצים שכל היישומים יפעלו יחד בסביבת הביצוע של OS/2, יוסבו רוב היישומים הפופולריים לסביבת הביצוע של OS/2.

המאפיינים של המישק לתכנות יישומים

המישק לתכנות יישומים ב-OS/2 פועל הן באצווה והן במקוון. היישום מפיק קריאה רחוקה (far call) לשם של פונקציה. אחר-כך הוא מעביר את כל הפרמטרים לפונקציה על-ידי דחיפתם לתוך **מחסנית** (stack). ניתן גם לדחוף את כתובות הפרמטרים לתוך המחסנית. כל הכתובות מורכבות **מבורר** (selector) בן 16 סיביות (סגמנט) ומערך בן 16 סיביות של **כתובת יחסית** (offset) בסגמנט (ר' הסקירה על המיקרו-מעבד בפרק זה). מספר הפרמטרים לפונקציה מסוימת הוא תמיד קבוע.

בחזרה מהפונקציה, ימצא קוד החזרה באוגר AX (register) (ר' הסקירה על המיקרו-מעבד בפרק זה). פרמטרים יכולים להיות מוחזרים על-ידי הפונקציה. במקרה זה, היישום ידחוף את הכתובת של הפרמטר המוחזר לתוך המחסנית לפני הקריאה לפונקציה. הפונקציה תמיד מוציאה את כל הפרמטרים מהמחסנית. הפונקציה שומרת לעצמה את כל האוגרים פרט לאוגרים AX ו-FLAGS.

לא נציג בספר זה כל פרט על כל מישק לתכנות יישומים. אתה יכול להשיג את המידע המפורט מתוך הספרות הטכנית הקיימת על OS/2. בפועל, כאשר אנו מכסים נושא מסוים, אנו דנים ברמות שונות של פירוט ברוב הפונקציות החשובות של המישק לתכנות יישומים הרלוונטיות לנושא זה. במספר מקרים, אנו דנים בפרמטרים ובפונקציות ברמה מאוד כללית של פירוט. במקרים אחרים, אנו מביאים תיאורים לוגיים של כל הפרמטרים לפונקציה מסוימת, כדי להראות מה מייצגים הפרמטרים. במקרים אחרים, אנו מראים את כל הפרמטרים לפונקציה מסוימת עד רמת פירוט של שפת מכונה. אנו מספקים את הפרטים אודות מישק התכנות בצורה סלקטיבית זו, כדי להצביע על הנקודות החשובות ולהקל על ההבנה של רמת התיפקוד של אפשרות מסוימת ב-OS/2.

סקירה על המיקרו-מעבד 80286

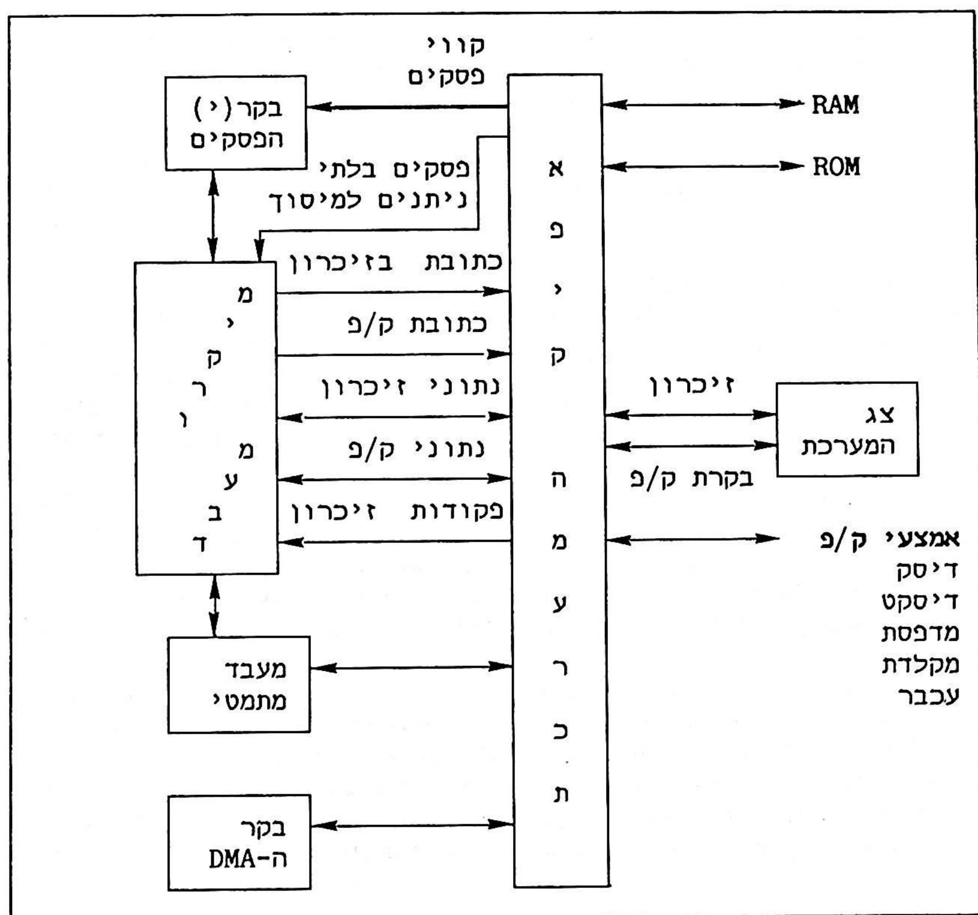
בסעיף זה אנו סוקרים את המאפיינים והפונקציות החשובות ביותר של המיקרו-מעבד 80286. סקירה זו תעזור לך להבין טוב יותר את התמיכה העומדת מאחורי חלק מהמאפיינים החדשים של OS/2. OS/2 נשענת על המאפיינים החדשים של 80286 שיספקו פונקציות כמו:

- תמיכה בזיכרון אמיתי גדול.
- תמיכה בזיכרון בפועל.
- הגנה על נתוני מערכת ההפעלה והיישום.
- הגנה מפני ביצוע לא נכון של פקודות היישום על-ידי מערכת ההפעלה.
- הגנה של המערכת מפני יישומים המבצעים פקודות מסוימות של המעבד.

לפני שנרד לפרטים של ה-80286, הבה נראה בקצרה כיצד נראית מערכת מחשב אישית.

מערכת המחשב האישי

תרשים 1 מכיל תיאור סכימטי פשוט וכללי של מערכת מחשב אישית. אנו רואים שהמיקרו-מעבד מהווה רק חלק קטן של מערכת מחשב אישית. אפיק המערכת הינו "האוטוסטרדה" שדרכה מתקשרים רכיבי מערכת המחשב האישי. זיכרון המערכת מכיל את הפקודות והנתונים שאליהם ניגש המיקרו-מעבד בזמן ביצוע. אמצעי הקלט/פלט מתקשרים ישירות למיקרו-מעבד, או משתמשים ב-DMA כדי להציב מידע בזיכרון המערכת. עכשיו, הבה נסתכל בכל אחד מהרכיבים האלה ביתר הרחבה ונכיר את הטרמינולוגיה שלהם.



תרשים 1. תיאור סכימטי של מערכת מחשב אישי

אפיק המערכת

אפיק המערכת (system bus) הינו מעין "משפך", שדרכו עוברת כל התקשורת בין המרכיבים השונים של המערכת. אם אי פעם התקנת כרטיסי הרחבה במחשב האישי שלך, חיברת את התקעים של כרטיסי ההרחבה לאפיק המערכת. האפיק מגדיר מערכת של חוקים שחלקי המערכת השונים חייבים לציית לה כך שיוכלו לעבוד במשולב כראוי. למאפיינים של אפיק המערכת יש השפעה חשובה על ההתנהגות והביצועים של המערכת. זו הסיבה שיבמ הפכה את **ארכיטקטורת המיקרו-ערוץ** (Micro Channel) לחלק אינטגרלי של ההכרזה על החומרה של **המערכת האישית/2** (PS/2 - Personal System/2).

זיכרון המערכת

זיכרון הוא חלק אינטגרלי של המערכת, ותפקידו לאחסן את הנתונים השוטפים ואת פקודות המעבד. המקום בזיכרון שמשתמשים בו נבחר על-ידי **כתובת פיסית** (physical address) שמוצבת באפיק המערכת. הגודל של הכתובת הפיסית שניתן להציב באפיק קובע את הכמות המקסימלית של זיכרון פיסי שניתן להתקין במערכת. ישנם שלושה סוגים שונים של זיכרון הקיימים במערכת מחשב אישית:

- Random Access Memory - RAM - זיכרון לגישה אקראית.
- Read Only Memory - ROM - זיכרון לקריאה בלבד.
- Memory Mapped I/O - קלט/פלט ממופה זיכרון.

ה-RAM נמחק כאשר מנתקים את המתח מהמערכת. ניתן לקרוא נתונים מה-RAM וניתן לכתוב בו. פקודות תכנית המרכיבות את היישום שנבחר על-ידי המשתמש נקראות לתוך ה-RAM על-ידי מערכת ההפעלה מאמצעי ק/פ אחר ומבוצעות על-ידי המיקרו-מעבד (על-ידי קריאת הפקודות מה-RAM). דוגמאות של אמצעי ק/פ המאחסנים פקודות של תכניות הם דיסקטים ודיסקים. רוב הזיכרון המרכיב את המערכת הוא RAM.

ה-ROM אינו מאבד לעולם את תכולתו, גם כאשר מנתקים את המתח למערכת. אולם, ה-ROM אינו ניתן לשינוי על-ידי המעבד, אשר יכול לקרוא ממנו בלבד. ה-ROM ב-PS/2 מכיל **פקודות** BIOS ונתונים של המערכת. BIOS הוא קיצור של Basic Input Output System - מערכת ק/פ בסיסית. ה-BIOS כולל פקודות מכונה ונתונים, המאפשרות למעבד לבצע פעולות בחומרת המערכת מבלי להכנס ישירות לחומרה. הדבר מאפשר למערכת ההפעלה להיות מוגנת משינויים מסוימים בחומרה. ב-PS/2 יש BIOS תואם, הנקרא **Compatibility BIOS**, מאותו הסוג של BIOS הנמצא ב-PC AT. BIOS זה תומך במצב האמיתי של ה-80286. BIOS מתקדם חדש הנקרא **Advanced BIOS** פותח עבור המערכת OS/2. BIOS זה תומך במערכת עם מאפיינים של ביצועים רב-משימתיים ותומך במצב המוגן של ה-80286. OS/2 וה-device drivers משתמשות ב-Advance BIOS ב-PS/2. פקודות ה-BIOS נמצאות במרחב הכתובות הפיסי בין 640KB ל-1MB.

ישנם אמצעי ק/פ חיצוניים אשר יש בהם RAM. RAM זה ידוע בשם ק/פ ממופה זיכרון. דוגמה טובה לכך הינה צגי המחשב האישי של יבמ. המסך שהמשתמש רואה הוא ייצוג ישיר של אותו חלק ב-RAM המהווה חלק אינטגרלי של הצג. כשתכנית משנה את זיכרון הצג, דברים שונים מופיעים על המסך. המעבד משנה את מצבי הבקרה של הצג כדי לשלוט על הדרך שבה יתורגם ה-RAM בצג להצגה סופית על המסך. RAM זה נמצא בד"כ במרחב הכתובות הפיסי בין 640KB ל-1MB.

אמצעי ק/פ חיצוניים

הגישה לאמצעי ק/פ חיצוניים נעשית בד"כ באמצעות כתובות ק/פ ולא באמצעות כתובות זיכרון. למיקרו מעבד יש פקודות שונות הגורמות להפקת כתובות הק/פ ולהעברת נתוני הק/פ. כתובת ק/פ לא תאפשר גישה לזיכרון וכתובת בזיכרון לא תאפשר גישה לכתובת ק/פ. המערכת מבינה את ההבדל בין מיעון זיכרון לבין מיעון של ק/פ, כי לאפיק המערכת יש קווי בקרה מיוחדים המראים אם הפעולה שמתבצעת הינה פעולת זיכרון או פעולת ק/פ.

באמצעי ק/פ חיצוניים יש בד"כ קבוצות של אוגרי בקרה הממופים לקבוצה של כתובות ק/פ. כאשר המעבד רוצה שאמצעי ק/פ חיצוני יבצע פעולה כלשהי, הוא יוציא הוראה לכתובת הק/פ המתאימה. מידע זה יוצב באוגר הבקרה של אמצעי הק/פ ובסופו של דבר יגרום לאמצעי הק/פ לבצע את הפעולה הדרושה. נתונים רגילים יכולים גם הם להיות מועברים לאמצעי הק/פ בדרך זו. גם אוגרי הבקרה של אמצעי הק/פ ניתנים לקריאה בידי המעבד כדי לקבל את המצב השוטף של אמצעי הק/פ. לדוגמה, המעבד יכול לקרוא את אוגר הבקרה בכניסה של המדפסת המקבילית, כדי לראות אם המדפסת מוכנה לקבל תו נוסף. כאשר התשובה חיובית, המעבד שולח את התו לכניסת ק/פ שונה, המכילה את אוגר הנתונים של המדפסת המקבילית.

אמצעי ק/פ מיוחדים הם אמצעי ק/פ לטיפול בגושי (blocks) נתונים. זאת אומרת, שהם מטפלים בחבילות של נתונים בפעולה אחת, במקום לטפל בתו (byte) אחד בלבד. לדוגמה, הדיסק ב-PC AT או ב-PS/2 מבצע את פעולות הכתיבה והקריאה עם 512 בתים בכל פעולה. מאחר ואין במעבד פקודת ק/פ האומרת "קרא 512 בתים", המעבד חייב לבצע מספר פקודות ק/פ לקריאת בית אחד או שנים בכל פעולה. אם אמצעי הק/פ משתמש **בגישה ישירה לזיכרון** (Direct Memmory Access - DMA) כדי להעביר נתונים לזיכרון המערכת, המעבד אינו צריך להיות מעורב בזמן פעולת העברת הנתונים.

גישה ישירה לזיכרון (DMA)

התקנים מסוימים משתמשים בבקר המערכת לגישה ישירה (Direct Memory Access - DMA) כדי להעביר נתונים לזיכרון המערכת וממנו. לכן המעבד אינו צריך להמשיך להוציא פקודות ק/פ לאמצעי הק/פ עבור כל בית או שני בתים של נתונים שמועברים. מה שקורה בפועל הוא, שהמעבד פוקד על בקר ה-DMA להעביר את הנתונים לכתובת פיסית מסוימת או ממנה, ופוקד על אמצעי הק/פ להתחיל להעביר את הנתונים. המעבד יכול לבדוק מדי פעם ולראות מתי הפעולה הסתיימה. ניתן גם לתכנת את אמצעי הק/פ שיפיק **פסק** (interrupt) כאשר הפעולה הסתיימה.

הפעולה המתוארת לעיל משתמשת **בערוץ DMA** יחיד. ברוב המערכות יש יותר מאשר ערוץ DMA אחד. אמצעי ק/פ לטיפול בגושים ואמצעים להעברת תו אחד בו-זמנית יכולים להיות מתוכננים כך שישתמשו ב-DMA. דוגמה לאמצעי העברת תו אחד בו-זמנית שתוכנן להשתמש ב-DMA הוא התקן ה-SDLC. הנתונים מגיעים בצורה של תו אחד בכל פעם, אבל המעבד אינו מעורב בהעברת כל תו של נתונים, מכיון שההתקן אוגר את התווים לקבוצות ואז מודיע למעבד שהם מוכנים להעברה. דיסק ודיסקטים הינם דוגמאות של אמצעים לטיפול בגושי נתונים היכולים להשתמש ב-DMA.

ב-PC AT, נתוני הדיסקטים מועברים בעזרת DMA, אבל נתוני הדיסק הקשיח

מועברים על-ידי סידרה של פקודות ק/פ שמופקות על-ידי המעבד. ב-PS/2, נתוני הדיסקט והדיסק הקשיח מועברים בערוצים נפרדים של ה-DMA.

כאשר אתה משתמש ב-DOS, שבו רק פעולה אחת מתבצעת בו-זמנית, אין שום חשיבות לכך שהפעולה נעשתה עם DMA. המעבד מחכה בכל מקרה שפעולת ה-DMA תסתיים, מכיון שאין לו ממילא עבודה אחרת לעשות. בסביבת OS/2, העובדה שהמעבד אינו כבול בזמן העברת הנתונים, יכולה לתרום לשיפור הנצילות שלו. לדוגמה, ב-PS/2 יכולים להעביר נתונים בשני ערוצי DMA נפרדים בזמן שהמעבד מבצע פעולה אחרת.

לא ניתן לקבל דבר מבלי לתת תמורה. אם תחזור לתרשים 1, תוכל לראות איך כל הנתונים חייבים להיות מועברים דרך אותו "משפך" הקרוי אפיק המערכת. אם יותר מדי העברות נתונים מתבצעות בו-זמנית, או אם העברה יחידה של נתונים משתמשת בחלק גדול מדי של **רוחב הפס של האפיק** (bus bandwidth) לפרק זמן ארוך מדי, אז פעולות ה-DMA יפריעו זו לזו או למעבד. המעבד חייב להגיע לזיכרון המערכת אפילו כאשר הוא צריך להביא את הפקודה הבאה בתור לעיבוד. אם המעבד אינו יכול להגיע לזיכרון המערכת כדי להביא הוראות, הוא נעצר ומחכה לאפיק שיתפנה. רוחב הפס של האפיק מגדיר את כמות הנתונים ביחידת זמן שהאפיק יכול להעביר. ככל שרוחב הפס יותר גדול, יותר נתונים יוכלו לעבור באפיק ביחידת זמן אחת, ויותר פעילות DMA תתבצע ללא הפרעה לפעילות המעבד. זהו שיקול מאוד חשוב כאשר מתכננים מתאמי חומרה המשתמשים ב-DMA בסביבה רבת משימות כמו ה-OS/2.

פסקים

התקנים מודיעים למיקרו-מעבד שהם רוצים להפיק "פסק" על-ידי הצבת אות בקווי אפיק המערכת המיועדים לכך. בקר הפסקים מתרגם את האותות האלה ומפיק פסק שבסופו של דבר מגיע למיקרו-מעבד. בקר הפסקים תומך בקווים רבים המעבירים פסקי חומרה באפיק המערכת ומתרגם את מה שעובר באפיק הפסקים לאותות מיוחדים ה"מובנים" למיקרו-מעבד. כל קו פסקים במערכת מתאים לרמת פסקים מסוימת, ברמת עדיפות אשר נקבעת לו. בקר הפסקים מטפל גם במתן העדיפויות לקווי הפסקים.

כאשר בקר הפסקים מנסה להפיק פסק למיקרו-מעבד, המיקרו-מעבד יגיב אם הוא מסוגל לקבל את הפסק. המיקרו-מעבד יפסיק את את העיבוד של זרם הפקודות המבוצע באותו רגע ויתחיל לבצע זרם אחר של פקודות. אנו נדון בדרך שבה נקבע זרם פקודות חדש בשלב יותר מאוחר של פרק זה.

זרם הפקודות החדש נקרא **שיגרת טיפול בפסקים** (interrupt handler). שיגרה זו נקבעת על-פי רמת הסיווג של הפסק שהופק, אשר תואם בד"כ לאמצעי ק/פ נתון. אי לכך, שיגרת הטיפול בפסקים מותאמת לטפל בפסקים השייכים לאמצעי ק/פ מסוים.

לדוגמה, אפשר לקבוע במדפסת המקבילית שהיא תפיק פסק בכל פעם שהיא תהיה מוכנה לקבל תו חדש להדפסה. ה-device driver ישלח תו לאוגר הנתונים של המדפסת המקבילית ולאחר מכן יחזיר את השליטה למערכת ויאפשר לה ביצוע עבודה אחרת מועילה יותר. לאחר שהתו הודפס, תפיק המדפסת המקבילית פסק. בקצב הדפסה של 40 תווים לשניה, זה יהיה לאחר 25 milisec מסיום הדפסה של כל תו. שיגרת הטיפול בפסקים של ה-device driver של המדפסת תשיג את השליטה הודות לפסק. היא תבדוק את הכניסה (port) של המדפסת המקבילית

כדי לקבוע את הסיבה לפסק, תראה שהמדפסת מוכנה לקבל תו נוסף, ותשלח אותו לכניסה לשם הדפסתו.

פסקים הם **מופעלי סף** (edge triggered) ב-PC AT **ורגישים לרמת-סיווג** (level sensitive) ב-PS/2. זהו תיאור של הדרך שבה מופיע אות הפסק באפיק המערכת ולאיזה סוג של אות פסק בקר הפסקים צריך לצפות. OS/2 תומכת רק באמצעי ק/פ אחד ברמת פסקים מסוימת בסביבת PC AT. OS/2 תומכת באמצעי ק/פ רבים באותה רמת פסקים במחשבי PS/2. בסביבת הפסקים הרגישים לרמת-סיווג, אמצעי הק/פ אינו מפסיק להפיק את אות האירוע לבקר הפסקים עד אשר אמצעי הק/פ מקבל שירות. בסביבת הפסקים מופעלי-סף, אמצעי הק/פ מפיק פולס פסק אחד שבקר הפסקים מגלה. אם המערכת בוחרת לאתחל את בקר הפסקים מבלי לשרת את אמצעי הק/פ, הפסק יאבד ואמצעי הק/פ לעולם לא יקבל שירות.

מעבד מתמטי

המעבד המתמטי הנוסף מספק סיוע חיוני למיקרו-מעבד. הוא מאפשר למערכת לעבד סוגים מסוימים של פקודות (כמו פעולות מתמטיות עם מספרים המיוצגים בשיטת הנקודה הצפה) במהירות גבוהה יותר מאשר אם היו מדמים אותן בעזרת סידרה של פעולות מעבד רגילות. המערכת יכולה להיות מכוונת כך שתגלה שהמעבד המתמטי לא קיים, ואז היא תדמה את הפקודות האלה ותבצע אותן באמצעות המיקרו-מעבד הראשי.

המיקרו-מעבד

המיקרו-מעבד מביא פקודות מזיכרון המערכת ומבצע אותן לפי הסדר שבו הן מאוחסנות בזיכרון. במהלך הביצוע של פקודות אלו, המעבד יכול לכתוב נתונים בזיכרון, לקרוא נתונים מהזיכרון, לכתוב נתונים לכתובת ק/פ, או לקרוא נתונים מכתובת ק/פ. פקודות מסוימות (כמו הסתעפות מותנה) יכולות לגרום למעבד לבצע פקודה אחרת מאשר הפקודה העוקבת.

בתוך המיקרו-מעבד יש זיכרון פנימי קטן, אשר מכיל את **האוגרים** (registers) של המיקרו-מעבד. פקודות המשתמשות בנתונים הנמצאים באוגרי המעבד הן בד"כ יותר יעילות ואינן דורשות מהמיקרו-מעבד לצאת לאפיק המערכת כדי לקרוא או לכתוב נתונים בזיכרון. ישנם בתוך המיקרו-מעבד אוגרים פנימיים נוספים שאינם משמשים לביצוע רגיל של פקודות, אלא עוזרים למיקרו-מעבד לתפקד.

מאפיינים רבים של מיקרו-מעבד מסוים מבדילים אותו ממיקרו-מעבדים אחרים, כמו:

- מספר האוגרים לשימוש כללי הנמצאים בו והגודל שלהם (מספר סיביות).
- העוצמה של מערכת הפקודות וכמה סיביות של נתונים הן יכולות לעבד בפעם אחת.
- הכמות הפיסית של נתונים שהמעבד יכול להעביר בבת-אחת לאפיק המערכת (מספר הסיביות).
- הדרכים השונות שבהן יכול המעבד למען את הנתונים שלו והאם הן ניתנות לשימוש על-ידי רוב הפקודות.

- האם ישנו מנגנון הגנה לשימוש מערכת ההפעלה ומהו.
- מבנה הפסקים של המעבד.

עכשיו, כשראינו כבר את המבנה הכללי של המיקרו-מעבדים, הבה נביט מקרוב ב-80286.

תיאור המבנה והתפקוד של ה-80286

הכרה של הזיכרון המקומי ב-80286 ואיך הוא מנוצל היא המפתח להבנת פעולותיו של ה-80286. רובו של הזיכרון המקומי ב-80286 מורכב מאוגרים. התכולה של האוגרים מוגדרת כ"מצב" שה-80286 נמצא בו. על-ידי ניתוח מצב נוכחי נוכל לקבוע בדיוק מהי הפעולה הבאה שה-80286 עומד לבצע.

הבט על תרשים 2. הוא מתאר את כל האוגרים ומשתנים אחרים המתארים מצב שבו אנו מתעניינים כדי שנוכל להבין את ה-80286. לפני שנחקור לעומק, נלמד תחילה את שיטת המיעון של ה-80286. כל כתובת מורכבת משני ערכים בני 16 סיביות:

- הראשון מציין את אחד **מאוגרי הסגמנט** (segment registers).
- השני מציין את **הכתובת היחסית** (offset) בתוך הסגמנט.

<u>סגמנטים לשימוש כללי</u>

- ארבע אוגרים בני 16 סיביות. כל אוגר יכול לשמש גם כ-2 אוגרים בני 8 סיביות.

AX = AH | AL
BX = BH | BL
CX = CH | CL
DX = DH | DL

- ארבע אוגרים נוספים בני 16 סיביות.

Base Pointer - BP (מצביע הבסיס)
Source Index - SI (אינדקס המקור)
Destination Index - DI (אינדקס היעד)
Stack Pointer - SP (מצביע המחסנית)

<u>אוגרי הסגמנטים</u>

ארבעה אוגרים בני 16 סיביות.

CS - Code Segment (סגמנט פקודה)
DS - Data Segment (סגמנט נתונים)
ES - Extra Segment (סגמנט נוסף)
SS - Stack Segment (סגמנט מחסנית)

<u>אוגרים מיוחדים</u>

F - Flags (דגלים)
IP - Instruction Pointer (מחוון פקודה)
MSW - Machine Status Word (מלת מצב מכונה)

<u>מידע חשוב אחר</u>

- אוגרים למתארים של ה-cache memory.
- GDT - Global Descriptor Table (טבלה למתארים גלובליים)
- LDT - Local Descriptor Table (טבלה למתארים מקומיים)
- IDT - Interrupt Descriptor Table (טבלה למתארים לפסקים)
- אוגר משימה, מצב משימה בסגמנט.

תרשים 2. תיאור פשטני של מבנה ה-80286

ערך אוגר הסגמנט מגדיר את **הסגמנט** (segment) שאליו אנו מתייחסים. ערך הכתובת מגדיר את הכתובת היחסית בסגמנט. ב-80286, יכולים להכנס בסגמנט עד 64KB (65,536 בתים - הכתובת המקסימלית היכולה להכנס באוגר בן 16 סיביות: $65,535 = 2^{16} - 1$). נדון להלן כיצד מתורגמים הערך באוגר הסגמנט והערך של הכתובת היחסית, במטרה ליצור את כתובת הזיכרון הפיסית.

איך יודע המעבד מהיכן להביא את הפקודה הבאה בזיכרון? הוא מביא את הפקודה הבאה על-ידי שימוש ב-CS כערך של אוגר הסגמנט וב-IP כערך של הכתובת היחסית. פקודות מסוימות (כמו הסתעפות מותנית) משנות את הערך של IP. אם לא נעשה שינוי ב-IP על-ידי הפקודה הנוכחית, אז הגודל של הפקודה הנוכחית מתוסף ל-IP וכתוצאה מכך תתבצע הפקודה הבאה בתור. אם נעשה שינוי ב-IP, אז פקודה אחרת מתוך הסגמנט המוגדרת על-ידי ה-CS תתבצע. פקודות מסוימות משנות הן את ה-CS והן את ה-IP. הדבר גורם להבאת פקודה הנמצאת בסגמנט אחר.

המחסנית (stack) הינה קטע מיוחד בזיכרון שמנוצל לאחסון מידע בזמן שקורה אחד מהאירועים הבאים:

- שיגרה אחת (שילוב של מספר פקודות) קוראת לשיגרה אחרת.
- יש צורך בשטח אחסון זמני והתכנית אינה רוצה להקצות במפורש שטח לנתונים בשביל אותו אחסון זמני.

סגמנט המחסנית הנוכחי מוגדר על-ידי אוגר SS. אוגרים SP ו-BP מגדירים בד"כ את הכתובת היחסית שמשתמשים בה (תלויה במטרה שלשמה נעשה שימוש במחסנית).

המעבד מקבל את הנתונים שלו מסגמנט המוגדר על-ידי אוגר הסגמנט DS או ES. הכתובת היחסית שלהם יכולה להיות שילוב של דברים רבים ושונים. הכתובת הפיסית יכולה להיות מקודדת במפורש לתוך הפקודה, או שהיא יכולה להיות שילוב אריתמטי כלשהו של אחד או שניים מתוך האוגרים לשימוש כללי, או ערך מפורש בן 16 סיביות. יש מספר הגבלות על סוגי האוגרים לשימוש כללי שניתן להשתמש בהם בכל מצב מסוים.

לרוב הפקודות יש מקור לנתונים ויעד שאליו הם נשלחים. מקור המידע יכול להיות מוגדר במפורש בפקודה, יכול להיות באוגר לשימוש כללי, או יכול להיות בזיכרון. היעד של הנתונים יכול להיות אחד האוגרים לשימוש כללי או מקום כלשהו בזיכרון. רוב הפקודות אינן יכולות לפעול בו-זמנית על מקור הנתונים ועל היעד שאליו הם נשלחים. אולם, פקודות מסוימות תוכננו במיוחד לעשות זאת. פקודות אלו משתמשות ב-SI וב-DI עבור הכתובות היחסיות של המקור והיעד, והן משתמשות ב-DS וב-ES כסגמנטים של המקור והיעד.

אוגר **הדגלים** (flags) מכיל מידע על התוצאה של הפעולה האחרונה (כמו סיבית הסימן, נגררת, גלישה וכו') ואם מתאפשרים פסקים במערכת. **מילת מצב מכונה** (machine status word) מכילה מידע על המעבד כמו, אם הוא צריך להמצא במצב אמיתי או במצב מוגן. הדיון על שני האוגרים האלה הינו מחוץ להיקף של ספר זה.

הערכים המוצבים באוגרים אלה (ובזיכרון מקומי אחר) בזמן הפעלת המעבד קובעים מה המעבד יעשה בזמן האתחול (reset). לאחר שמאתחלים את ה-80286, הוא מועבר למצב אמיתי על-ידי הצבת הערכים F000H ב-CS ו-FFF0H ב-IP

(שים לב, "H" מסמל ייצוג הקסה-דצימלי). כפי שנראה מיד, הדבר גורם למעבד להתחיל להביא פקודות מכתובת פיסית FFFF0H, קצת מתחת ל-1MB. שלושת אוגרי הסגמנט האחרים מאופסים.

מיעון ב-80286 - מצב אמיתי לעומת מצב מוגן

לכתובות הפיסיות המופקות על-ידי ה-80286 יש שני מרכיבים: ערך בן 16 סיביות באוגר הסגמנט וערך בן 16 סיביות המציין את הכתובת היחסית בתוך הסגמנט. הכתובת היחסית נקבעת על-ידי משתנים המקבלים ערכים שונים בהתאם למצב.

שני הערכים בני 16 הסיביות מעובדים בצורה נפרדת על-ידי ה-80286 בהתאם למצב העבודה של המעבד - אמיתי או מוגן. כאשר ה-80286 נמצא במצב אמיתי, הוא מפיק כתובות פיסיות בצורה זהה למיקרו-מעבד 8088 או 8086. הבה נתבונן בתרשים 3 כדי להבין איך מופקות פקודות פיסיות במצב האמיתי של ה-80286.

הפקת כתובת פיסית במצב אמיתי

התרכז בתרשים 3 ולמד אותו. לדרך זו של הפקת כתובות פיסיות יש מספר מאפיינים חשובים ביותר. דנו כבר בצורה שבה קובע אוגר הסגמנט את הסגמנט שבו תהיה הכתובת. מכיון שהכתובת היחסית היא ערך בן 16 סיביות, נובע שכל סגמנט הוא בן 64KB (65,536 בתים). ערך הכתובת היחסית נע בין 0 ל-65,536. סגמנטים תמיד מתחילים בכתובת פיסית שהיא כפולה של 16. הכתובת הפיסית הסופית שמופקת היא ערך בן 20 סיביות, כך שרק ל-1MB (00000H-FFFFFH) של זיכרון ניתן להכנס כשמשתמשים במצב האמיתי של ה-80286.

האם שמת לב למאפיינים החשובים מכל? על-ידי ידיעת הערך הנמצא באוגר הסגמנט, אנו יודעים בדיוק היכן ניתן למצוא את הסגמנט בזיכרון. מכיון שתכנית היישום מציבה ערכים באוגר הסגמנט, היא יכולה לקבוע לאיזה זיכרון פיסי (כתובת) היא תכנס. זו הסיבה מדוע נאמר קודם שבמצב האמיתי של ה-80286 הכתובת בפועל זהה לכתובת האמיתית. זכור, הכתובת בפועל היא הכתובת שהתכנית רואה והכתובת האמיתית הינה הכתובת הפיסית המשמשת לגישה לזיכרון הפיסי במערכת.

יישומים יכולים גם להניח שאם הם יוסיפו 1 לערך הנמצא באוגר הסגמנט, הם יוכלו להכנס לאותו זיכרון, כאילו הכפילו את הערך של הכתובת היחסית פי 16 (בהנחה שאין גלישה).

אלו הן הנחות שתכניות בסביבת DOS עושות כל הזמן. זו אחת הסיבות מדוע תכניות אלה אינן יכולות להתבצע בצורה ישירה בסביבת המצב המוגן של ה-80286.

1. קח אוגר סגמנט בן 16 סיביות (ערך: 65,535 - 0, או: 0000H - FFFFH) והכפל אותו ב-16. במלים אחרות, הפוך אותו לערך בן 20 סיביות על-ידי כך, שתשים ארבעה אפסים בסיביות הנמוכות.
2. הערך שיצא מאוגר הסגמנט הוא בתחום: 00000H - FFFFFH ‏(0 - 1,048,650). כיון שערך ארבע הסיביות הנמוכים הוא תמיד אפס, הערך יכול להיות רק כפולות של 16 בתוך אותו טווח.
3. הוסף ערך כתובת יחסית בן 16 סיביות אל הערך בן 20 הסיביות.
4. התוצאה היא כתובת פיסית סופית בת 20 סיביות הנעה בין הערכים 00000H ל-FFFFFH (אין מתייחסים לגלישה).

A.	אוגר הסגמנט	ערך אוגר סגמנט בן 16 סיביות
B.	0000	ייצוג בעזרת ערך בן 20 סיביות
	+	
C.	כתובת יחסית	ערך בן 16 סיביות של כתובת יחסית
D.	כתובת פיסית	ערך כתובת פיסית בן 20 סיביות

תרשים 3. תהליך הפקת כתובת פיסית במצב אמיתי

הפקת כתובת פיסית במצב מוגן

עד עתה קיבלנו מושג כללי לגבי המצב המוגן. דנו כבר בעובדות הבאות הקשורות להפקת כתובות במצב המוגן של ה-80286:

- תמיכה בזיכרון פיסי במערכת, אשר גודלו עד 16MB.
- הכתובות בפועל לא זהות לכתובות האמיתיות (הפיסיות). הערך באוגר הסגמנט אינו תואם בצורה ישירה לכתובת הפיסית.
- יש תמיכה בכתובות בפועל שאינן נמצאות כרגע בזיכרון (ולכן יש צורך להביא אותן לזיכרון).
- ניתן לתמוך במרחבי כתובות בפועל שונים לתכניות יישום שונות.
- קיימת הגנה בזיכרון בין מערכת ההפעלה ליישומים.

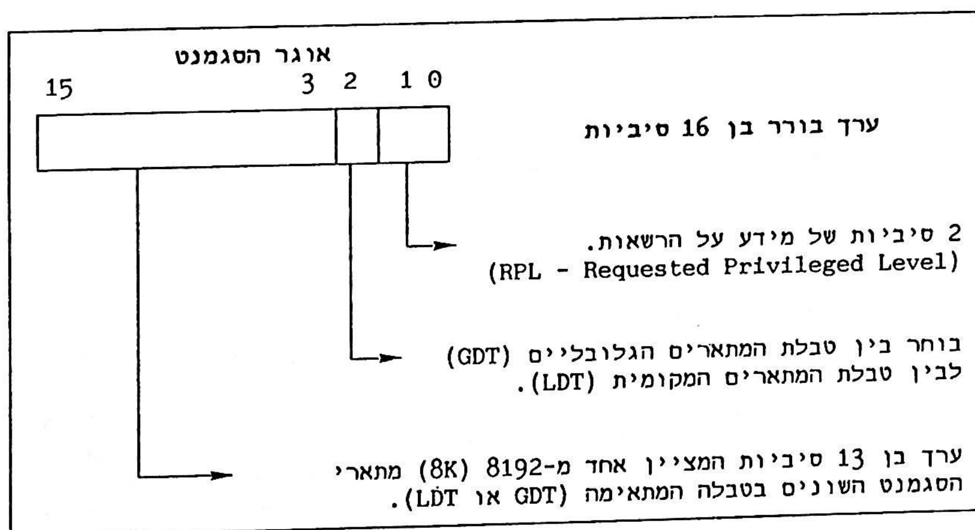
הרמז לגבי צורת העבודה של המצב המוגן הוא שהערך באוגר הסגמנט אינו קובע את הבסיס לחישוב הכתובת הפיסית של הסגמנט כפי שקורה במצב האמיתי. הערך באוגר הסגמנט, כאשר ה-80286 נמצא במצב מוגן, נקרא **בורר** (selector). הבורר פועל כאינדקס לטבלה מיוחדת, שיש בה את כל המידע על

סגמנט הזיכרון הפיסי. מידע זה נקרא **מתאר הסגמנט** (segment descriptor). הבט על תרשים 4 ותראה כיצד מתורגם ערך הבורר. שתי הסיביות הנמוכות ביותר אינן משמשות בתהליך הפקת הכתובת הפיסית, כי יש להם תפקיד במנגנון ההגנה. תפקיד זה ידון בהמשך.

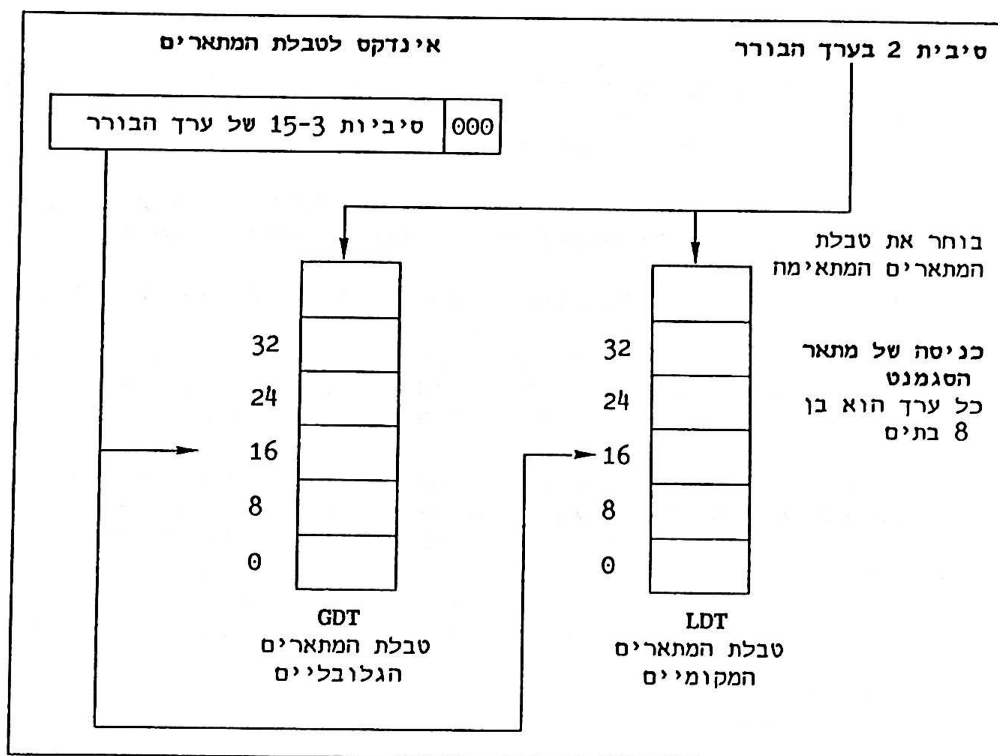
יש שני סוגים שונים של טבלאות במערכת המכילות מתארי סגמנטים. הטבלאות הן: **טבלת המתארים הגלובליים** (Global Descriptor Table - GDT) **וטבלת המתארים המקומיים** (Local Descriptor Table - LDT). סיבית מס' 2 בערך הבורר קובעת את סוג הטבלה שאליה פונים לצורך בחירת מתאר הסגמנט. זכור, מתאר הסגמנט מכיל את כל המידע המתאר את הזיכרון הפיסי שאנו מעוניינים בו.

תרשים 5 מתאר כיצד נבחר מתאר הסגמנט. לאחר שנבחרה הטבלה המתאימה, סיביות 3-15 של ערך הבורר מאפשרות לבחור אחד מתוך 8,192 מתארי הסגמנטים השונים. כל מתאר הוא בן 8 בתים.

טבלת המתארים הגלובליים (GDT) מכילה בד"כ סגמנטים שמשתמשים בהם במערכת בלי קשר ליישום שמופעל. לדוגמה, פקודות מערכת ההפעלה ומתארי הסגמנטים של הנתונים נמצאים ב-GDT. קבוצות של פקודות הצריכות להיות מוגנות זו מזו מציבות את מתארי הסגמנטים שלהן בטבלאות LDT שונות. מערכת ההפעלה שומרת LDT שונה לכל קבוצה של פקודות הדורשת שהזיכרון שלה יהיה נפרד מקבוצות אחרות של פקודות. ב-OS/2, לכל תהליך יש LDT הקשור אליו.



תרשים 4. תרגום ערך הבורר



תרשים 5. בחירת מתאר הסגמנט

הבט בתרשים 6 וראה את פריטי המידע הנמצאים במתאר הסגמנט אודות כל סגמנט. המידע שאנו מעונינים בו כרגע הוא הכתובת הפיסית הבסיסית בת 24 סיביות. ערך זה בן 24 סיביות קובע היכן מתחיל הסגמנט בזיכרון הפיסי. ערך הכתובת היחסית בן 16 הסיביות מתוסף אל הכתובת הפיסית שקיבלנו ממתאר הסגמנט, כדי לקבוע את הכתובת האמיתית הסופית. בהנחה שסגמנט גדל עם הזמן (ולא קטן כסגמנט המחסנית), הכתובת היחסית עלולה לעבור את גודל הסגמנט ואז המעבד ינסה להכנס לזיכרון שאינו חלק מהסגמנט. זהו מצב שגיאה המצביע על כך שהתכנית טעתה. ישנה אפשרות למצב שגיאה אחר, שבו תכנית תנסה לעקוף את אמצעי ההגנה ולהכנס לזיכרון. הבורר ומתאר הסגמנט מכילים מידע המאפשר הגנת סגמנטים.

דומה שכל הבדיקות האלו המתבצעות בכל פעם שנכנסים לכתובת, מעמיסות מאוד על המעבד. אם נחזור לתרשים 2, נוכל לראות למה משמשים אוגרי המתארים של ה-cache. המעבד שומר את המידע בזיכרון המקומי שלו, הקיים במתארי הסגמנט הנוכחיים שנבחרו על-ידי ערכי הבורר הנמצאים כרגע באוגרי הסגמנטים. המעבד אינו צריך להכנס שוב ל-GDT או ל-LDT, אלא אם השתנה אוגר הסגמנט. זו הסיבה מדוע שינוי באוגר הסגמנט במצב מוגן דורש הרבה משאבים במונחים של ביצוע. המעבד שומר גם את הערך הנוכחי של הכתובת ואת הגודל של ה-GDT וה-LDT בזיכרון המקומי שלו.

1. ערך בן 24 סיביות הקובע את הכתובת הבסיסית של הסגמנט הפיסי. מאפשר גישה ל-16MB של זיכרון פיסי אמיתי.

2. ערך בן 16 סיביות הקובע את גודל הסגמנט (עד 64KB).

3. האם הסגמנט נמצא כרגע בזיכרון הפיסי.

4. האם ערך הבורר המתאים למתאר סגמנט זה הוטען לתוך אוגר הסגמנט. הדבר מציין סימן שבוצעה כניסה לסגמנט זה.

5. האם זה סגמנט פקודה, סגמנט נתונים, או סגמנט "מיוחד" (*).

6. אם הסגמנט הינו סגמנט המכיל פקודות אז:
   - האם המעבד צריך, או לא צריך, לקרוא נתונים מסגמנט זה.
   - האם סגמנט הפקודות הזה ערוך כהלכה (*).

7. אם הסגמנט הינו סגמנט המכיל נתונים:
   - האם המעבד צריך, או לא צריך, לכתוב לתוך סגמנט זה.
   - האם הערך הגבולי תואם לגודל הסגמנט.

8. מידע על הרשאות - רמת הרשאת המתאר (Descriptor Privilege Level - DPL)

(*) מחוץ להיקף של ספר זה.

תרשים 6. המידע הנמצא באוגר הסגמנט

חזור לרשימת המאפיינים של תהליך מיעון הכתובות במצב מוגן המתוארים בתחילת סעיף זה. אנו רואים איך ניתן לתמוך ב-16MB של זיכרון אמיתי, וכיצד מרחבים שונים של כתובות בפועל נתמכים בעזרת שימוש בטבלאות LDT שונות. הכתובת שהתכנית מציבה באוגר הסגמנט אינה תואמת לחלוטין לכתובת האמיתית שמשתמשים בה.

אם נצרף את המידע שהוצג בתרשים 6, נוכל לראות שאנו תומכים בכתובות בפועל שאינן נמצאות למעשה בזיכרון. אם הסגמנט אינו נמצא כרגע בזיכרון האמיתי, אז מתאר הסגמנט שלו יודיע שהוא אינו נמצא. אם התכנית מטעינה לאוגר הסגמנט ערך בורר המתאים לסגמנט שאינו נמצא, המעבד יעבור למצב של שגיאה לגבי פעולה זו ויטפל בשגיאה בעזרת שיגרה מיוחדת של מערכת ההפעלה. שיגרה זו תמצא היכן מאוחסן הסגמנט על הדיסק, תעביר אותו לזיכרון האמיתי, ותעדכן את מתאר הסגמנט כדי שיראה היכן נמצא הסגמנט בזיכרון האמיתי. אם המעבד יצטרך לפנות מקום לסגמנט, יתכן שהוא יעביר סגמנט אחר לדיסק. הסגמנט שיועבר החוצה לדיסק נקבע על-ידי מערכת ההפעלה, והוא יהיה בד"כ הסגמנט שהיה הכי פחות בשימוש לאחרונה. המידע על הגישה לסגמנטים הנמצא בטבלאות המתארים עוזר למערכת ההפעלה לקבוע איזה מהסגמנטים היה הכי פחות בשימוש לאחרונה.

עוד הבדל חשוב בין מצב אמיתי למצב מוגן הוא, שבמצב מוגן לכל סגמנט יש ציון נפרד של גודל. במצב אמיתי, כל הסגמנטים היו בעלי גודל של 64KB.

במצב מוגן, התכניות מקצות סגמנטים שגודלם נגזר מצרכי התכנית. ובכך הן חוסכות בזיכרון אמיתי וזיכרון בפועל הנמצא על הדיסק. אם תכנית טועה ומנסה להכנס לזיכרון הגדול מזה של הסגמנט השייך לה, תופק הודעת שגיאה ומערכת ההפעלה תבין שמשהו אינו כשורה.

עוד לא השלמנו את הדיון בהגנת הזיכרון. אחד מהאספקטים החשובים ביותר של הגנת הזיכרון הוא רמת הרשאה, אשר תידון בסעיף הבא.

הגנה והרשאה

במצב המוגן של ה-80286 ניתן למצוא מאפייני אבטחה ומתן הרשאות. במצב האמיתי של המעבד אין כל אמצעי הגנה שהוא. זאת אומרת, שכל תכנית יכולה להכנס לכל מקום בזיכרון הפיסי (עד ל-1MB) ויכולה לבצע כל פקודת מעבד אשר מותרת במצב אמיתי.

מערכת ההפעלה מבטיחה שכל יישום יהיה מוגן מיישומים אחרים על-ידי כך שלכל יישום נשמרות טבלאות LDT משלו. כל ערך בורר שהיישום יפיק אינו יכול להגיע למתאר סגמנט שאינו נמצא בטבלת ה-LDT הנוכחית.

בתרשים 6 מתוארים מנגנוני הגנה הנבדקים בכל פעם שמבוצעת גישה לזיכרון. בכל פעם שמופר כלל המשמש להגנה, המערכת מוציאה הודעת **חריגה** (exception), אשר גורמת למעבד להתחיל לבצע שיגרה לטיפול בשגיאות חריגה (exception handler). זהו מצב דומה מאוד לפסק, פרט לכך שהמעבד עשה משהו שגרם לחריגה, ואילו פסק מופק בד"כ על-ידי התקן חיצוני. מערכת ההפעלה היא זו שמטפלת בחריגה. המעבד מתוכנן כך שיאפשר לפקודה שגרמה את החריגה להתבצע מחדש לאחר שמצב החריגה במעבד אותחל (reset).

להלן מספר דוגמאות של פעולות של מיעון כתובות הגורמות למצבי חריגה ומצביעות על כך שקיימת בעיה בתכנית:

- הבורר מצביע על מתאר סגמנט הנמצא מעבר לתחום הנוכחי של טבלת המתארים.
- הכתובת היחסית עוברת את הגודל הנוכחי של הסגמנט.
- הפרה של כללי ההרשאה.
- CS (אוגר סגמנט הפקודה) הוטען בערך בורר התואם למתאר סגמנט נתונים. המעבד אינו יכול לבצע פקודות מתוך סגמנט נתונים.
- המעבד מנסה לכתוב לתוך סגמנט נתונים המוגדר לקריאה בלבד.
- המעבד מנסה לקרוא מתוך סגמנט המכיל פקודות לביצוע בלבד.

מצב שגיאה יווצר גם כאשר המעבד מנסה להטעין אוגר סגמנט בבורר, אשר מתאים לסגמנט שאינו נמצא בזיכרון. מערכת ההפעלה תמצא את הסגמנט על הדיסק, תעביר אותו לזיכרון האמיתי, תשנה את מתאר הסגמנט כך שיודיע שהסגמנט נמצא במקום פיסי מסוים בזיכרון, ותחדש את הביצוע בפקודה שגרמה לשגיאה.

ללא מנגנון ההרשאות, היה קשה מאוד לבצע בצורה יעילה את תכנית ההגנה הזאת. לדוגמה, תכנית יודעת שעל-ידי הצבת ערך מסוים בסיבית 2 של ערך הבורר ניתן להגיע ל-GDT. מאחר וה-GDT מכיל סגמנטים רבים של מערכת ההפעלה, התכנית יכולה לנסות להשתמש בערכי בורר של ה-GDT, אשר אין להם שום קשר לתכנית ולנתוניה. איך החומרה מכוונת למנוע מתכניות לבצע פעולות כאלה?

יש ארבע רמות הרשאה ב-80286. הן ממוספרות מ-0 עד 3, כאשר 0 היא רמת ההרשאה הגבוהה ביותר ו-3 - רמת ההרשאה הנמוכה ביותר. מערכת ההפעלה מנהלת את ההרשאות. ב-OS/2, לרוב היישומים יש רמת הרשאה 3, ולגרעין (kernel) מערכת ההפעלה יש רמת הרשאה 0. ניתן לכתוב יישומים שיקבלו הרשאת ק/פ (I/O Privilege - IOPL). אנו נדון ב-IOPL מאוחר יותר. ב-OS/2, כאשר ליישום יש הרשאת ק/פ, הוא נמצא ברמת הרשאה 2 ואינו יכול לבצע את כל הפעולות שהיו מותרות לו ברמת הרשאה 3.

הגישה העומדת מאחורי הרשאות אומרת, שפקודה המתבצעת במחשב יכולה להכנס לסגמנט נתונים רק כאשר רמת ההרשאה שלו שווה לשלה, או נמוכה יותר. הדבר מונע מיישומים להכנס לנתוני מערכת ההפעלה, אבל מאפשר למערכת ההפעלה להגיע לנתוני היישומים שהיא צריכה. רמת ההרשאה של הפקודות שרצות כרגע נקראת **רמת הרשאה נוכחית** (Current Privilege Level - CPL) והיא שוות-ערך לשתי הסיביות הנמוכות באוגר הסגמנט CS. רמת ההרשאה של הנתונים שנכנסים אליהם מוגדרת על-ידי שתי סיביות במתאר הסגמנט, ונקראת **רמת הרשאת המתאר** (Descriptor Privilege Level - DPL).

סביבת הביצוע יכולה להיות מכוונת כך שיישום לא יוכל לשנות את ה-CPL שלו. מערכת ההפעלה תוכל להגן על עצמה ללא קושי אם תגדיר ליישום CPL=3, ותגדיר DPL=0 לכל הנתונים החשובים שלה (בדוגמה זו גם רמות 1 ו-2 יספיקו).

הסתעפויות רחוקות וקריאות יוכלו להיות לסגמנט פקודה בעל רמת הרשאה זהה. בכך, נמנע מיישומים לקרוא לסגמנטים המכילים פקודות של מערכת ההפעלה. ישנם מתארי סגמנטים מיוחדים הנקראים **שערים** (gates), אשר משתמשים בהם לצורך העברת השליטה בין רמות הרשאה שונות ולצורך שינוי רמת ההרשאה הנוכחית. פרטים אלה הם מחוץ להיקף של ספר זה.

מהי הרשאת ק/פ (IOPL)? מערכת ההפעלה קובעת באילו רמות הרשאה מותר לבצע פקודות מסוימות הקשורות לפעולות ק/פ. אלה הן הפקודות:

- פקודות המאפשרות למעבד לקרוא נתונים ממרחב הכתובות של הק/פ, או לכתוב בו.
- פקודות שמאפשרות, או שאינן מאפשרות, למעבד לקבל פסקים מהתקנים חיצוניים.
- פקודת LOCK (נעל את האפיק. מחוץ להיקף של ספר זה).

לפקודות אלו יש בדיקות מיוחדות הקשורות אליהן, מכיון שהן יכולות לגרום נזק למערכת אם הן מבוצעות בתכנית שלא קיבלה הרשאה. על-ידי דרישה מיישום לבקש IOPL, מערכת ההפעלה יכולה להגביל את מספר תכניות היישום היכולות לפגוע במערכת. מערכת ההפעלה יכולה להגדיר את הערך המקסימלי של CPL שיכול לקבל IOPL. ב-OS/2, רמת IOPL נקבעה ל-2. תכניות יישום רצות בד"כ עם CPL=3, ועל-כן הן אינן יכולות לקבל IOPL, אך ישנו API המאפשר זאת. המשתמש יכול להגדיר את ה-CONFIG.SYS בצורה כזו שתאפשר ליישומים לקבל IOPL. ישנן הגבלות על מה שיישומים יכולים לעשות עם IOPL בסביבת OS/2. לדוגמה, יישומים אינם יכולים לטפל בפסקים של התקנים חיצוניים מתוך סגמנט פקודה עם הרשאת ק/פ. השיטה המועדפת לטיפול בפסקים היא דרך device drivers.

ישנן פקודות נוספות הניתנות לביצוע רק מתוך רמת הרשאה 0 (CPL=0), שהיא רמת ההרשאה הגבוהה ביותר. פקודות אלו מגדירות את גודלן של טבלאות GDT ו-LDT ואת מקומן.

ישנן פקודות רבות שאינן קיימות ב-8088 וקיימות ב-80286. רוב הפקודות האלו קיימות גם במצב האמיתי ב-80286. ראינו כמה עשירה המערכת של מנגנוני ההגנה הנמצאת ב-80286.

טיפול בפסקים

כבר הצגנו מדוע התקנים חיצוניים צריכים להפיק פסקים. יש להסביר עתה כיצד נקבע איזו שיגרה תקבל את השליטה לאחר שהמעבד קיבל את הפסק. אחד הפסקים שהמערכת יכולה לקבל נקרא **פסק בלתי ניתן למיסוך** (Non-Maskable Interrupt - NMI). המעבד אינו יכול להתעלם מ-NMI, מכיון שהוא מודיע למעבד על שיבוש חמור במערכת. דוגמה טובה לכך תהיה תקלה בזיכרון.

ישנם חמישה-עשר ערוצי חומרה ב-PC AT וב-PS/2 המשמשים להעברת פסקים, אשר הטיפול בהם נעשה בידי שני בקרי פסקים המחוברים ביניהם. כאשר OS/2 מאתחלת את בקרי הפסקים, היא בוחרת איזה פסקים יועברו בכל אחד מחמישה-עשר ערוצי החומרה. מספר הפסק המתאים לערוץ חומרה מסוים קובע איזו שיגרה תקבל שליטה לאחר שהמעבד יקבל את הפסק.

בקר הפסקים מפקח על חמישה-עשר ערוצי חומרה המשמשים להעברת פסקים, והוא אחראי גם על הפקת מספר הפסק הנכון במעבד. בקר הפסקים יפיק את המספר המתאים בהתאם לכללי סיווג הפסקים שלו. אפשר לשלול את יכולת המעבד לקבל פסקים חיצוניים מבקר הפסקים בכל רגע, על-ידי ביצוע פקודה מתאימה. הפקודות המאפשרות למעבד לקבל פסקים ואלה השוללות יכולת זו, יכולות להתבצע תחת IOPL, או ברמה גבוהה יותר.

איך נעשית ההתאמה בין מספר הפסק לשיגרה המקבלת את השליטה לאחר שהמעבד קיבל את הפסק? כדי לענות על השאלה הזו נצטרך להציג במעבד טבלה נוספת - **טבלת מתארי הפסקים** (Interrupt Descriptor Table - IDT). הטבלה מכילה 256 ערכים הדומים מאוד לערכי מתארי הסגמנט שכבר ראינו. ההבדל הגדול ביותר ביניהם הוא בכך שאלה הם מתארי סגמנט מיוחדים הנקראים **שערים** (gates). כל שער מצביע על שיגרת הטיפול בפסקים המתאימה לאותו מספר הפסק. הדיון בשערים הינו מעבר להיקף של ספר זה. נאמר אך זאת: כאשר דנו בהרשאות לפקודות, הזכרנו ששערים משמשים להעברת השליטה על הביצוע מרמת הרשאה אחת לאחרת.

הפיסקה הקודמת מתארת כיצד נבחרת במצב מוגן השיגרה המתאימה לטיפול בפסקים. במצב אמיתי התהליך של בחירת שיגרת הטיפול בפסקים דומה, פרט לכך שנעשה שימוש בסוג שונה של IDT. טבלת IDT זו שונה בגודלה מזו שבמצב המוגן בנוסף לכך שהערכים שלה במצב אמיתי אינם שערים. למעשה, כל ערך בטבלת ה-IDT הוא כתובת לפי מצב אמיתי, המתבססת על ערך סגמנט בן 16 סיביות וכתובת יחסית בת 16 סיביות. ערך זה תואם לערך הכתובת הפיסית במצב אמיתי, אשר תקבל את השליטה כאשר יופק מספר הפסק המתאים. מספר הפסק מתאים לערך מסוים בטבלת ה-IDT השייכת למצב האמיתי. יישומים בסביבת DOS המופעלים ב-OS/2 צריכים להשתמש בשירותי ה-DOS כדי לשנות את שיגרת הטיפול בפסקים למספר פסק מסוים. הם אינם צריכים לשנות בצורה ישירה את טבלת ה-IDT השייכת למצב האמיתי.

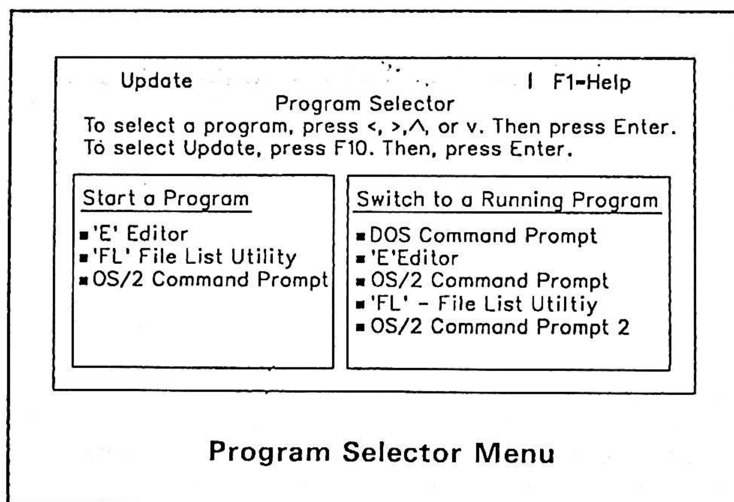
מספר הפסק גורם למעבד להעביר את השליטה לשיגרה המתוארת על-ידי ערך ה-IDT המתאים למספר הפסק. משתמשים במחסניות כדי להבטיח שפקודה שאיבדה את השליטה תוכל להתבצע בשלב מאוחר יותר. ישנם סוגים רבים של שערים

שמשמשים להשיג סוגים שונים של דברים. פסקים בלתי ניתנים למיסוך (NMI) גורמים להפקת מספר פסק 2.

תוכנה יכולה להפיק פסקים על-ידי ביצוע פקודת INT במעבד. OS/2 אינה תומכת ביישומים המשתמשים בפקודת INT במצב מוגן. גם מצבי שגיאה יוצרים פסקים שהמעבד אינו יכול להתעלם מהם. מערכת ההפעלה חייבת לטפל במצבי השגיאה.

סיכום

כיסינו כמות רבה של חומר בפרק זה. כרגע צריכה להיות לך הבנה טובה מאוד של הפונקציות הכלליות של OS/2 ורקע בסיסי על הדרך שבה עובד ה-80286. זה הזמן להתחיל להכנס לפרטים ולהסביר כיצד להשתמש במקצת מהפונקציות שדנו בהם. בפרק הבא נדון באפשרויות השימוש בזיכרון שמספקת OS/2 ואיך תכניות יכולות להשתמש באפשרויות אלה באמצעות המישק לתכנות יישומים (API).



Program Selector Menu

בורר התכניות

פרק 2

ניהול הזיכרון

לזיכרון המערכת תפקיד חשוב ביותר בפעולתה של מערכת המחשב. פקודות מערכת ההפעלה ונתוניה וכן גם תכניות היישום ונתוניהן, נמצאות בזיכרון המערכת. המיקרו-מעבד לוקח את הפקודות מזיכרון המערכת ומבצע אותן. לשם ביצוע פקודות שונות פונה המיקרו-מעבד לזיכרון המערכת לשם הבאת הנתונים.

אחד השיפורים המשמעותיים ש-OS/2 מספקת לעומת ה-DOS, הוא בכך שהיא מנצלת אפשרויות טיפול נוספות בזיכרון כפי שמאפשר המיקרו-מעבד 80286. בפרק הקודם סקרנו בפירוט את היתרונות שיש ל-80286 בהשוואה ל-8086 מבחינת ניצול זיכרון המערכת.

נסכם ונאמר שה-8086 פונה לכתובת בזיכרון הפיסי באופן ישיר מכתובת שהיישום מציין. כתובת זו היא ייצוג ישיר של הכתובת הפיסית ומבוססת על סגמנט בן 16 סיביות וכתובת יחסית בת 16 סיביות. ערך הסגמנט בן 16 סיביות מגדיר סגמנט בן 64KB בזיכרון הפיסי, והכתובת היחסית מגדירה לאיזו כתובת בתוך הסגמנט יש לפנות. בעזרת 8086 ניתן להכנס עד ל-1MB של זיכרון פיסי.

הכתובת שהיישום משתמש בה ב-80286 אינה ייצוג ישיר של הזיכרון הפיסי. זו הסיבה מדוע היא נקראת **כתובת בפועל** (virtual address). הכתובת מתבססת על ערך בורר בן 16 סיביות וכתובת יחסית בת 16 סיביות. ערך הבורר משמש להצבעה על מתאר סגמנט מתוך טבלה הנמצאת בזיכרון. המתאר מכיל את פרטי המידע החשובים על סגמנט הזיכרון הפיסי וכולל את הדברים הבאים:

- המקום בזיכרון הפיסי שבו נמצא הסגמנט (אפשרות גישה בזיכרון הפיסי עד ל-16MB), או מידע שהסגמנט אינו נמצא בזיכרון.
- גודל הסגמנט (הסגמנטים אינם חייבים להיות בגודל 64KB).
- מידע על הגנה, הרשאה וסוג השימוש (לדוגמה, קריאה בלבד).

יתרונות עיקריים

כתוצאה מניצול אפשרויות הטיפול בזיכרון שמאפשר ה-80286, יכולה OS/2 לספק את השיפורים הבאים:

- תומכת במערכות שבהן עד 16MB של זיכרון פיסי. מערכת DOS מספקת 640KB של זיכרון לשימוש ישיר של היישומים, אך ב-OS/2, מערכת ההפעלה והיישומים יכולים לבצע פקודות ולהביא נתונים ישירות מזיכרון פיסי שיכול להגיע עד 16MB. מרחב הכתובות בין 640KB ל-1MB מנוצל על-ידי המערכת ל-BIOS ולק/פ ממופה זיכרון.

- תומכת ביישומים המשתמשים בכמות זיכרון הגדולה מ-640KB ואף גדולה מכמות הזיכרון הפיסי במערכת. ניתן לכתוב יישומים שישתמשו ביותר מ-640KB. בנוסף לכך, OS/2 מאפשרת לסגמנטים של זיכרון לא להיות בזיכרון הפיסי. בפועל, הסגמנט נשמר על הדיסק ומובא לזיכרון כאשר יש צורך להשתמש בו, בנוהל של תמיכה בזיכרון-יתר.
- תומכת ביישומים רבים המשתמשים בו-זמנית בזיכרון הגדול מהזיכרון הפיסי במערכת. בנוסף לכך, משתמשים בטבלאות שונות של מתארים מקומיים (LDT), כדי לאפשר ליישומים להיות מוגנים זה מזה.
- OS/2 עצמה משתמשת באפשרויות החדשות שמספק ה-80286. שירותים אלה מאפשרים הגנה על מערכת ההפעלה מפגיעה על-ידי יישומים, והם מאפשרים לפקודות מערכת ההפעלה ולנתונים שלה להשתתף בתהליך התמיכה בזיכרון-יתר. OS/2 אינה צורכת כמות כה גדולה של זיכרון כפי שהיה נדרש, אילו כל התוכנה היתה צריכה להמצא באופן קבוע בזיכרון הפיסי.

OS/2 מספקת הגנה בין יישומים שונים ובין היישומים לבין מערכת ההפעלה. בנוסף לכך, היא מאפשרת למשתמש להריץ יישום או קבוצה של יישומים המשתמשים בכמות גדולה של זיכרון. ניתן להשתמש ביותר מ-640KB זיכרון, ואף ביותר זיכרון מאשר ניתן למצוא פיסית במערכת.

אפשרויות השימוש בזיכרון גדול מתורגמות בצורה ישירה ליתרונות משמעותיים למשתמש. יישומים יכולים לעשות את הדברים הבאים:

- לפתור בעיות יותר מתוחכמות.
- להיות קלים יותר לשימוש.
- לטפל ביותר נתונים.
- לספק משוב חזותי מפורט יותר.

יישומים יכולים להציע את היתרונות האלה אם הם לא יוגבלו על-ידי זיכרון פיסי קטן מדי, או על-ידי מרחב כתובות קטן מדי במערכת. אבל יותר חשוב, המשתמש יכול להריץ יותר מיישום אחד בו-זמנית מבלי לדאוג שיחסר לו זיכרון פיסי במערכת. מערכות תוכנה ויישומים שונים יכולים להיות בשימוש מבלי שכל פקודה וכל נתון יימצאו בזיכרון הפיסי.

כמו בכל דבר אחר, חייבים גם פה להתחשב בחלופות, הן ביחס לזמן הביצוע והן ביחס לביצועים עצמם. יישומים צריכים להיבנות כך, שהם יצמצמו את העברת הנתונים בין הזיכרון הפיסי והדיסק. המשתמש חייב להתקין זיכרון פיסי גדול במידה מספקת כדי לאפשר זמן תגובה סביר בסביבת הביצוע הרצויה.

תמיכה בזיכרון-יתר

ניתן להבין בקלות את מהות התמיכה בזיכרון פיסי עד 16MB. הסידור של מתאר הסגמנט מאפשר הפקת כתובת פיסית עד 16MB.

אולם, כאשר אנו אומרים שיותר זיכרון יכול להיות בשימוש מאשר באמת קיים במערכת, למה אנו מתכוונים? כאשר פונים לכתובת ב-80286, יכול לקרות שמתאר הסגמנט יודיע שהסגמנט לא נמצא בזיכרון. הכתובת המייצגת תופעה זו נראית כמו כל כתובת אחרת הנמצאת בשימוש מערכת ההפעלה והיישומים. רק

כאשר המיקרו-מעבד מנסה לקבוע היכן הכתובת נמצאת בזיכרון הפיסי, מתברר שהכתובת אינה נמצאת הזיכרון הפיסי באותו זמן. הדבר גורם למעבד להפיק הודעת שגיאה האומרת שהסגמנט לא נמצא. מערכת ההפעלה חייבת לטפל בשגיאה זו ולהבין היכן נמצא סגמנט הזיכרון הדרוש.

הסגמנטים "שאינם נמצאים בזיכרון" בזמן מסוים נשמרים בקובץ סגמנטים מועברים (swap file) הנמצא בדיסק הקשיח. התמיכה בזיכרון-יתר אפשרית על-ידי שימוש בשלוש טכניקות:

- העברת סגמנטים.
- ביטול סגמנטים.
- הזזת סגמנטים.

העברת סגמנטים (segment swapping) מתארת העברה של סגמנטים מהזיכרון הפיסי אל קובץ הסגמנטים הנמצא בדיסק. לשם כך משתמשים באלגוריתם הפועל על פי **עקרון** LRU (Least Recently Used), כדי לציין אילו סגמנטים היו בשימוש הכי פחות לאחרונה, כדי להעביר אותם מהזיכרון הפיסי אל הדיסק. כאשר צריך להביא סגמנט זיכרון מהדיסק, חייבים להקצות לו מקום בזיכרון הפיסי, ועל-כן יתכן ויהיה צורך להעביר סגמנט מהזיכרון הפיסי כדי לפנות עבורו את המקום הדרוש. OS/2 מנסה להמנע מהעברה לדיסק של סגמנטים שמשתמשים בהם בתדירות גבוהה ו/או לאחרונה. יש סיכוי גבוה שסגמנטים אלה יהיו בשימוש שוב, ובקרוב. אם סגמנט "פופולרי" יועבר לדיסק, יהיה צורך להחזיר אותו מיד לזיכרון הפיסי, בפעולה אשר תעמיס את המערכת ללא צורך ותשפיע על ביצועיה.

כדי לפנות מקום בזיכרון הפיסי לסגמנטים חדשים דרוש **ביטול סגמנטים** (segment discard). אין צורך להעביר לדיסק סגמנטים מכילי פקודות, כי הם אינם משתנים וניתן לבטל אותם בזיכרון. כאשר מגיע הזמן להשתמש בהם שוב, הם יובאו שוב מן המקור בדיסק. ניתן לבטל גם סגמנטים מסוימים המכילים נתונים. כאשר יוצרים סגמנט נתונים ניתן ליצור אותו כניתן לביטול. כאשר היישום משתמש בסגמנט, הוא "נועל" אותו כך שלא ניתן יהיה לבטל אותו. יישומים המאפשרים לבטל סגמנטים עוזרים למערכת ההפעלה לספק סביבה יעילה יותר של תמיכה בזיכרון-יתר.

נניח שצריך להביא מהדיסק סגמנט בן 64KB. לשם כך יש צורך בזיכרון פיסי **רצוף** בן 64KB שייתכן ולא יהיה פנוי. זו הנקודה שבה חשובה הטכניקה של הזזת סגמנטים. במקרה זה דרושה **הזזת סגמנטים** (segment motion) בזיכרון כדי לקבץ שטחים קטנים ולא מנוצלים לשטח פנוי אחד, שיהיה גדול ורציף.

בפרק 4 כללנו דוגמה של תכנית שבה נדרשת תמיכה בזיכרון יתר. בדוגמה זו, הצגנו את המישק לתכנות הזיכרון, הנקרא DosAlloeSeg. התכנית תוכננה כך שהאדם המריץ אותה יוכל לפרט את כמות הזיכרון שברצונו להקצות. בדרך זו אפשר לנסות ולהקצות כמויות שונות של זיכרון ולראות את ההבדלים בהתנהגות המערכת כתוצאה מכך. אנו מתארים את המישקים השונים לתכנות הזיכרון מאוחר יותר בפרק זה.

הגנת הזיכרון

יישומים (תהליכים) מוגנים זה מזה מאחר ויש להם טבלאות LDT שונות. OS/2 יוצרת טבלת LDT שונה לכל תהליך במערכת. החלק של ערך הבורר בכתובת משמש את היישום כאינדקס לטבלת LDT מסוימת לצורך בחירת מתאר הסגמנט. אין אפשרות שיישום אחד יקבל מתאר סגמנט השייך ליישום אחר, כי הוא אינו יכול לגרום ל-80286 להכנס לטבלת LDT של יישום אחר. לכן, יישומים אינם יכולים לקרוא או להרוס מרחבי כתובות של יישומים אחרים.

יישומים צריכים לפעמים להשתמש באותו חלק של הזיכרון. הדבר דרוש כאשר יישום אחד צריך ליצור קשר עם יישום אחר על-ידי שימוש בקטע משותף של הזיכרון. OS/2 מספקת אפשרות זו דרך המישק לתכנות השימוש בזיכרון. יישומים יכולים ליצור סגמנטים שניתן להשתמש בהם במשותף. סוגים מסוימים של סגמנטים משותפים בזיכרון הם **סגמנטים בעלי שמות משותפים** (-name shared segments). נסביר סגמנטים אלה בפרק זה, בסעיף העוסק במישק לתכנות יישומים.

יישומי OS/2 פועלים במחשב ברמת הרשאה 3. יישומי OS/2 שיש להם הרשאת ק/פ פועלים ברמת הרשאה 2 בזמן שיש להם הרשאת ק/פ (IOPL). OS/2 עצמה וה-device drivers פועלות ברמת הרשאה 0. רמת הרשאה 0 הינה רמת ההרשאה בעלת מספר הזכויות הרב ביותר. רמת הרשאה ב-80286 מאפשרת למערכת ההפעלה להיות מוגנת מיישומים ובנוסף לכך היא מאפשרת למערכת ההפעלה לדרוש מיישומים הצריכים הרשאת ק/פ למלא אחר חוקים מיוחדים.

מאפייני זיכרון המערכת

OS/2 מעמידה לרשות סביבת הביצוע של DOS את כמות הזיכרון המירבית. היא עושה זאת על-ידי כך שהיא משתמשת בחלק קטן בלבד של הזיכרון מתחת ל-640KB. החלק של OS/2 הנמצא דרך-קבע (resident) בזיכרון אינו ניתן להעברה החוצה אל הדיסק, וכן גם כל ה-device drivers במערכת, אשר נמצאים מתחת לגבול ה-640KB. גם הם אינם ניתנים להעברה החוצה. שאר שטח הזיכרון שמתחת ל-640KB ניתן לשימוש על-ידי סביבת הביצוע של DOS.

חלק נוסף של OS/2 נמצא דרך-קבע בזיכרון מעל כתובת הזיכרון המציינת את 1MB. פרט לחלקי ה-OS/2 הנמצאים דרך קבע בזיכרון, ישנו חלק של OS/2 הניתן להעברה, בדיוק כמו יישומים. הדבר מאפשר שימוש יותר יעיל בזיכרון הפיסי של המערכת.

טעינת סגמנט מראש וטעינה לפי דרישה

סגמנטים של יישומים מוגדרים **כנטענים מראש** (preload), או **כנטענים לפי דרישה** (load on demand). כאשר סגמנט נטען מראש, הוא נטען לזיכרון מיד לאחר שהיישום מתחיל לפעול. זאת אומרת, שהסגמנט ינסה להשתמש בזיכרון פיסי, אבל יתכן והוא יועבר החוצה אם אין מספיק זיכרון פיסי פנוי.

כאשר סגמנט מאופיין כנטען לפי דרישה, הוא אינו מוטען לזיכרון עד אשר

היישום צריך להשתמש בו (בנתונים או בפקודות). סגמנט הנטען לפי דרישה אינו מעמיס על זיכרון המערכת עד אשר צריך להשתמש בו. סגמנט הנטען לפי דרישה לא יתפוס לעולם זיכרון פיסי, או מקום בקובץ הסגמנטים המועברים. סגמנטים של יישומים שסביר להניח שלא יהיו בשימוש הם מועמדים טובים להקבע כסגמנטים הנטענים לפי דרישה.

השיטה לטעינת סגמנטים מראש ולטעינה לפי דרישה חייבת להשקל בתשומת לב רבה במטרה להפחית את העומס על המערכת. עומס זה יגדל אם היישומים ישתמשו בטעינת סגמנטים מראש.

מאפיינים הניתנים לקביעה על-ידי המשתמש

למשתמש יש אפשרות לקבוע רבים ממאפייני השימוש בזיכרון המערכת באמצעות פרמטרים בקובץ CONFIG.SYS.

המשתמש יכול להחליט שחלק מהזיכרון הפיסי לא יהיה זמין ליישומים, על-ידי קביעת דיסק בפועל או על-ידי קביעת זיכרון מטמון (cache). המשתמש חייב לשקול את היתרונות שבשימוש בדיסק בפועל, או בזיכרון מטמון, מול העומס שיווצר במערכת כאשר יהיה פחות זיכרון פיסי הזמין לשימוש של היישומים.

המשתמש יכול לבטל העברות ותזוזות של הסגמנטים במערכת. הוא ירצה לעשות זאת בסביבת יישומים מיוחדת, אשר גורם הזמן בה הוא קריטי. אפשר שיישום כזה לא יפעל כהלכה אם הוא יצטרך לחכות לפקודה, או לנתון, שאינם זמינים לגביו באופן מיידי, כיון שהם הועברו החוצה או הוזזו.

המשתמש יכול לקבוע שסביבת הביצוע של DOS תהיה קטנה יותר מהמקסימום שנקבע לה (קצת מתחת ל-640KB). המשתמש גם יכול לקבוע שסביבת הביצוע של DOS לא תהיה קיימת בכלל. משתמשים שאין להם צורך להריץ יישומים בסביבת הביצוע של DOS צריכים לוודא שלא תיווצר סביבה כזאת באופן אוטומטי. בכך יישאר יותר זיכרון פיסי פנוי עבור היישומים החשובים.

המשתמש יכול להחליט היכן יהיה קובץ הסגמנטים המועברים. אם קיימת אפשרות שקובץ הסגמנטים המועברים יגדל וישתמש בכל הדיסק, יתכן והוא ירצה לשים את הקובץ הזה במחיצה נפרדת בדיסק.

מאפייני סגמנטים ב-Module Definition File

תיאור של **קובץ להגדרת מודול** (Module Definition File - MDF), ניתן בפרק על גישות מתקדמות בתכנות (ראה פרק 7). מתכנת יישומים יוכל להשתמש ב-MDF לקביעת רבים ממאפייני הסגמנטים המכילים את פקודות היישום, או את

נתוניו. חלק ממאפייני סגמנטים אלה מובאים להלן:

- האם הסגמנט נטען מראש, או נטען לפי דרישה.
- אם זה סגמנט המכיל פקודות, האם הוא נקרא כנתונים ומבוצע, או שהוא מבוצע בלבד. אם מנסים לקרוא כנתונים סגמנט המוגדר לביצוע בלבד, נקבל הודעת שגיאה.
- אם זה סגמנט נתונים, האם יש צורך לקרוא ממנו ולכתוב עליו, או רק לקרוא ממנו. אם מנסים לכתוב לתוך סגמנט המוגדר בקריאה בלבד, ה-80286 יוציא הודעת שגיאה.

שיקולי ביצועים בתכנון יישומים

כפי שדנו קודם, יש לבנות את היישומים באופן כזה שכמות העברות והזזות הסגמנטים במערכת יהיה נמוך ככל האפשר. ניתן לעשות זאת על-ידי סידור הסגמנטים מכילי הנתונים והפקודות, כך שהפקודות והנתונים שהסיכוי שיהיו בשימוש הינו הגבוה ביותר יקובצו לסגמנטים משותפים. הפקודות והנתונים שהסיכוי שיהיו בשימוש הינו הנמוך ביותר (כמו עזרה, או עיבוד שגיאות) יקובצו בסגמנטים נפרדים.

הסגמנטים יוגדרו כנטענים מראש, או כנטענים לפי דרישה, בהתאם לסיכוי ולתכיפות השימוש בהם.

למתכנת היישומים חשוב גם לשקול את גודל הסגמנטים. סגמנטים גדולים טובים לנתונים ופקודות שישתמשו בהם בתדירות גבוהה. ככל שהסגמנט גדול יותר, כך גובר העומס על המערכת כתוצאה מהעברה או מהזזה שלו, וגם השטח שהוא תופס בזיכרון הפיסי גדול יותר. שילוב של פקודות ונתונים הנמצאים בשימוש תדיר עם כאלה שאינם נמצאים בשימוש תדיר, אינו רצוי וגם בזבזני מבחינת הזיכרון הפיסי במערכת.

יישומים צריכים להתחשב בשיקולים הבאים:

- גדלי הסגמנט.
- מספר הסגמנטים.
- מספר הסגמנטים שיהיו צריכים להיות נוכחים בזיכרון הפיסי בזמן הביצוע הרגיל של היישום.
- כמות ההעברות שתצטרך להיות במצב של תמיכה בזיכרון-יתר.

המתכנת צריך לדאוג לכך שהמשתמש הסופי יוכל להריץ מספר יישומים בו-זמנית. המתכנת אינו צריך לנסות לכתוב תכנית אופטימלית לסביבת הפעלה של יישום אחד. לדוגמה, אין להגדיר את כל הסגמנטים של היישומים כנטענים מראש, כדי לגרום לכך שהיישום ירוץ מהר יותר (אם למערכת יש זיכרון פיסי המספיק להרצת יישום אחד). המתכנת צריך לפתח את היישום כדי שיתנהג בצורה טובה בסביבה רבת יישומים. המשתמש יעריך יישומים שיעבדו בצורה אופטימלית בסביבה רבת יישומים, והוא לא יבחר ביישום שמפריע ללא צורך לביצועים של שאר היישומים במערכת.

מישקים לתכנות הזיכרון

אינך צריך להיות מתכנת יישומים כדי שתרצה להבין את אפשרויות ניהול הזיכרון שמספקת OS/2. אתה צריך להבין אפשרויות אלה כדי שתוכל לעשות את הדברים הבאים:

- לכונן את המערכת שלך כראוי (דרך CONFIG.SYS).
- להבין איך כמות הזיכרון הפיסי שיש לך משפיעה על ביצועי המערכת בזמן הרצת קבוצה כלשהי של יישומים.

אם אתה משתמש בשפות עיליות כדי לכתוב יישום פשוט, לא תהיה חייב להשתמש במישקים לתכנות הזיכרון ש-OS/2 מספקת. אולם על-ידי לימוד של האפשרויות שמספקים מישקים לתכנות הזיכרון, תבין טוב יותר איך יישומי OS/2 יכולים לנצל את אפשרויות הטיפול האלה בזיכרון גדול.

בדיון על המישקים לתכנות הזיכרון, השתמשנו במונח **תהליך** (process). בשלב זה, תהליך לגביך הוא דבר הזהה ליישום. זו יישות ב-OS/2 שבבעלותה נמצאים משאבי המערכת. אחד ממשאבים אלה הוא הזכות להכנס לסגמנט של הזיכרון. יישום יכול להיות מבוסס על תהליך אחד, או על מספר תהליכים.

מישקים לתכנות הזיכרון

OS/2 מאפשרת לתהליכים להקצות סגמנטים בזיכרון. כאשר מקצים סגמנט בעזרת המישק לתכנות יישומים (Application Programming Interface - API), מוחזר ערך בורר בן 16 סיביות לתהליך. התהליך משתמש בערך הבורר כדי להכנס לסגמנט בזיכרון. התהליך יכול ליצור סגמנטים בכל גודל עד 64KB. אם הוא רוצה, הוא יכול גם לשנות את גודל הסגמנט, דבר המאפשר לתהליך להשתמש בצורה יעילה יותר בזיכרון המערכת. בנוסף לכך, התהליך יכול להודיע ל-OS/2 שהוא אינו רוצה להשתמש יותר בסגמנטים בזיכרון. הדבר מאפשר ל-OS/2 לבטל את הערכים המתאימים בטבלאות ה-LDT ולהפסיק להעביר סגמנטים החוצה.

תהליכים יכולים ליצור סגמנטים שניתן להתחלק בהם עם תהליכים נוספים. אחת השיטות לעשות זאת גורמת לכך שייתנן "שם" לסגמנט המשותף, כדי לאפשר לתהליך אחר מיעון לסגמנט. שיטה אחרת מאפשרת לתהליך אחד להתיר במפורש לתהליך אחר לבקש להשתמש בסגמנט המשותף, ללא איזכור השם הלוגי שמזהה אותו.

תהליכים יכולים לייצור סגמנטים הניתנים לביטול על-ידי OS/2. כאשר התהליך רוצה להשתמש בסגמנט, הוא מודיע ל-OS/2 במפורש שלא תבטל את הסגמנט כרגע (נועל את הסגמנט). כאשר התהליך אינו צריך את הסגמנט לזמן מה, הוא מודיע ל-OS/2 שהסגמנט ניתן לביטול כעת (משחרר את הסגמנט). כאשר OS/2 צריכה לפנות מקום בזיכרון הפיסי, היא תשתמש באופציה זו ותבטל את הסגמנט. כאשר התהליך צריך להשתמש שוב בסגמנט, הוא מודיע ל-OS/2 שהסגמנט אינו ניתן לביטול (נעול) כעת. אם OS/2 ביטלה כבר את הסגמנט, ניתנת הודעה על כך לתהליך (החזרת מידע מפונקציית הנעילה) והתהליך צריך ליצור את הסגמנט מחדש. למרות שטכניקת הביטול נראית כעבודה נוספת, היא מאוד מועילה לתהליכים הדורשים כמות גדולה של זיכרון זמני. תהליכים אלה אינם רוצים להעמיס על המערכת את כל ההעברות שהיו נדרשות, אילו לא היו משתמשים בטכניקה זו.

אחד האספקטים המעניינים ביותר של הדרך, שבה תכניות DOS נכנסות לזיכרון הוא, שהכתובת הפיסית תואמת בצורה ישירה לכתובת הסגמנט בת 16 סיביות המשמשת את התהליך. עובדה זו מאפשרת לתהליכים, אשר יש להם צורך ביותר מ-64KB של זיכרון, לפנות לכתובות של מבני נתונים גדולים. בצורה זאת הם יכולים לעבור על כל מרחב הנתונים מבלי לחפש אחר כתובות חדשות של סגמנטים. OS/2 מספקת ליישומים שיטה להקצות סגמנטים רבים בני 64KB, אשר לכל אחד מהם יש ערכי בורר הנגזרים מזה של ערך הבורר הקודם. ערך הבורר הבא מתקבל על-ידי הוספה של ערך קבוע לערך הבורר הנוכחי. כך היישום יכול לבקש שטח גדול בזיכרון (יותר גדול מ-64KB) ולהכנס לכל השטח באופן ישיר על-ידי שימוש בערך הבורר של הסגמנט הראשון בלבד שמתאר את שטח הזיכרון הגדול.

OS/2 מאפשרת לתהליך ליצור ערך בורר לפקודה הנמצאת בשטח המכיל נתונים. בעזרת ערך בורר זה, התהליך יכול לבצע פקודה הנחשבת גם לנתון, אשר יתכן ונכתב על-ידי תהליך אחר.

ישנם מקרים שבהם תהליך צריך הרבה קטעים קטנים של זיכרון. אם התהליך בונה סגמנטים נפרדים לכל מספר קטן של בתים שהוא צריך, נוצר עומס רב על המערכת. OS/2 מספקת מנגנון שהתהליכים משתמשים בו כדי לפצל סגמנט אחד גדול למספר קטעים קטנים. OS/2 עוקבת אחר אותם חלקי זיכרון מתוך הסגמנט הגדול שאינם נמצאים בשימוש. התהליך מגיש בקשות לחלקים קטנים של זיכרון מתוך הסגמנט הגדול, ו-OS/2 מחזירה את הכתובת היחסית הנכונה בתוך הסגמנט שאיתה צריך התהליך להשתמש. מאפיין זה של ה-OS/2 נקרא **הקצאת-משנה לזיכרון** (Memory Suballocation), והוא מסופק כסידרה של שגרות קישור דינמיות.

ניהול הסגמנטים בזיכרון

מישק התכנות ב-OS/2 מספק מנגנון המאפשר ליצור סגמנטים, לשנות את גודלם ולשחרר אותם. אפשר להפוך סגמנט נתונים לסגמנט של פקודות הניתנות לביצוע. בהמשך נפרט את הפקודות המשמשות למטרות אלו.

DosAllocSeg יוצרת סגמנט בכל גודל עד 64KB. המישק מחזיר ערך בורר בן 16 סיביות שהתהליך משתמש בו כדי להכנס לסגמנט בזיכרון שזה עתה נוצר על-ידי המערכת. הסגמנט החדש ניתן להזזה ולהעברה החוצה. התהליך יכול לבקש שלסגמנט שנוצר יינתנו המאפיינים הבאים:

- הסגמנט יהיה משותף, על-ידי כך שהתהליך הנוכחי יאפשר לתהליך אחר לגשת אליו. ראה DosGiveSeg.
- הסגמנט יהיה משותף, על-ידי כך שתהליך אחר יצליח להכנס לסגמנט זה. ראה DosGetSeg.
- הסגמנט ניתן לביטול. זה יכול לקרות במערכת שבה מותקן זיכרון קטן.

DosReallocSeg משנה גודל של סגמנט לאחר שכבר נוצר, על-ידי העברה למישק של ערך הבורר והגודל הרצוי. אם הסגמנט הוא משותף, ניתן להגדיל את הסגמנט בלבד. אם הסגמנט מאופיין כניתן לביטול, בקשה זו מבצעת את אותו תפקיד כמו DosLockSeg.

DosFreeSeg מאפשרת לתהליך לשחרר סגמנט שנוצר קודם, או שהוא השיג כניסה אליו. אם הסגמנט הוא משותף, הוא יוקצה מחדש (ישוחרר) רק לאחר שכל התהליכים שהשיגו כניסה אליו או יצרו אותו יוציאו בקשה זו.

DosLockSeg תהליך יכול ליצור סגמנט הניתן לביטול. אם הוא עושה זאת, אז בכל פעם שהוא רוצה להשתמש בסגמנט (פרט לאחר שהסגמנט נוצר לראשונה) הוא חייב להשתמש במישק התכנות הזה. שימוש במישק מודיע למערכת שהיישום ישתמש כעת בסגמנט, והמערכת אינה יכולה לבטל אותו כעת. אם קריאה זו נכשלת, התהליך יודע שהסגמנט בוטל והוא חייב ליצור אותו מחדש עם .DosAllocSeg

DosUnlockSeg מודיעה למערכת שיש לה אישור לבטל את הסגמנט. אפשר לבצע קינון (nesting) של DosLockSeg ו-DosUnlockSeg. במלים אחרות, אם מבצעים N פעמים קריאות של DosLockSeg, חייבים לבצע גם N פעמים קריאות DosUnlockSeg לפני שהמערכת תוכל לבטל את הסגמנט. המערכת תוכל עדיין להזיז או להעביר סגמנט שננעל.

DosUnlockSeg מודיעה למערכת שהיא יכולה לבטל כעת סגמנט שמותר לביטול, אשר היה נעול עד לרגע זה. השיטה של ביטול סגמנטים היא שיטה שבה תהליכים מאפשרים למערכת להשתמש בצורה טובה יותר בזיכרון הפיסי הקיים. לדוגמה, אם תהליך משתמש בכמות גדולה של זיכרון זמני, זו יכולה להיות שיטה יעילה להפחית את ההשפעה של השימוש בכל הזיכרון הזה.

DosCreateCSAlias יוצרת ערך בורר תקף לסגמנט פקודה מתוך ערכי בורר של סגמנטים המכילים נתונים (DS, ES, או SS). הסגמנט חייב להיות בבעלות בלעדית (לא משותפת) של התהליך, ואינו יכול להיות חלק מגוש של זיכרון שהוקצה על-ידי DosAllocHuge. התהליך מטעין קבוצה של פקודות תקפות לתוך סגמנט נתונים ומשתמש בפונקציה זו כדי שהוא יוכל להתייחס לסגמנט זה כסגמנט פקודות.

סגמנטים משותפים

מישק התכנות מאפשר לתהליך שיצר את הסגמנט בזיכרון לשתף תהליכים אחרים בגישה אליו.

DosGiveSeg מתירה לתהליך שיצר את הסגמנט בזיכרון לאפשר לתהליך אחר להכנס לאותו סגמנט. כדי שבקשת DosGiveSeg תפעל בצורה נכונה, התהליך חייב להודיע למערכת במפורש בזמן יצירת הסגמנט, שהוא עומד לעשות זאת.

התהליך שיצר את הסגמנט עם DosAllocSeg מעביר למערכת את ערך הבורר של הסגמנט ואת המספר המזהה של התהליך (Process ID - ראה בפרק הבא) שישתמש במשותף בסגמנט. המערכת מחזירה את ערך הבורר שהתהליך האחר יצטרך להשתמש בו, כדי להכנס לאותו סגמנט. התהליך המקורי מעביר את ערך הבורר הזה לתהליך האחר על-ידי שימוש בצורה כלשהי של תקשורת בין תהליכים.

DosGetSeg משיגה גישה לסגמנט שנוצר על-ידי סגמנט אחר. התהליך שיצר את הסגמנט חייב להתיר לתהליך אחר לפנות לסגמנט בעזרת המישק הזה. התהליך האחר מעביר את ערך הבורר של הסגמנט הרצוי למישק הזה, והמערכת מתירה לתהליך האחר להכנס לסגמנט.

סגמנטים בעלי שמות משותפים

אפשר לתת שמות לוגיים לסגמנטים בזמן יצירתם. בכך, תהליכים אחרים במערכת יוכלו להכנס לסגמנט על-ידי שימוש בשם הסגמנט.

DosAllocShrSeg יוצרת סגמנטים בכל גודל עד ל-64KB, אשר מזוהים בעזרת שם. התהליך מעביר את הגודל והשם למישק. ערך הבורר שהתהליך צריך להשתמש כדי להכנס לסגמנט מוחזר למבקש.

השם שהתהליך נותן לסגמנט הוא באותו מבנה כמו שם קובץ ב-OS/2. השם חייב להיות בתת-מדריך \sharemem\, ולכן כל השמות יתחילו במחרוזת התווים \sharemem\. מישק DosAllocShrSeg מתיר לתהליך אחר כניסה לאותו סגמנט על-ידי שימוש בשם שניתן לסגמנט על-ידי התהליך שיצר אותו.

DosGetShrSeg משיגה אישור כניסה לסגמנט שנוצר עם פונקציית המערכת DosAllocShrSeg. כל התהליכים צריכים לדעת את השם שניתן לסגמנט בזמן שהוא נוצר. התהליך מעביר את השם של הסגמנט למערכת ומקבל בחזרה את ערך הבורר שישמש לו לכניסה לסגמנט.

ניהול סגמנטים גדולים

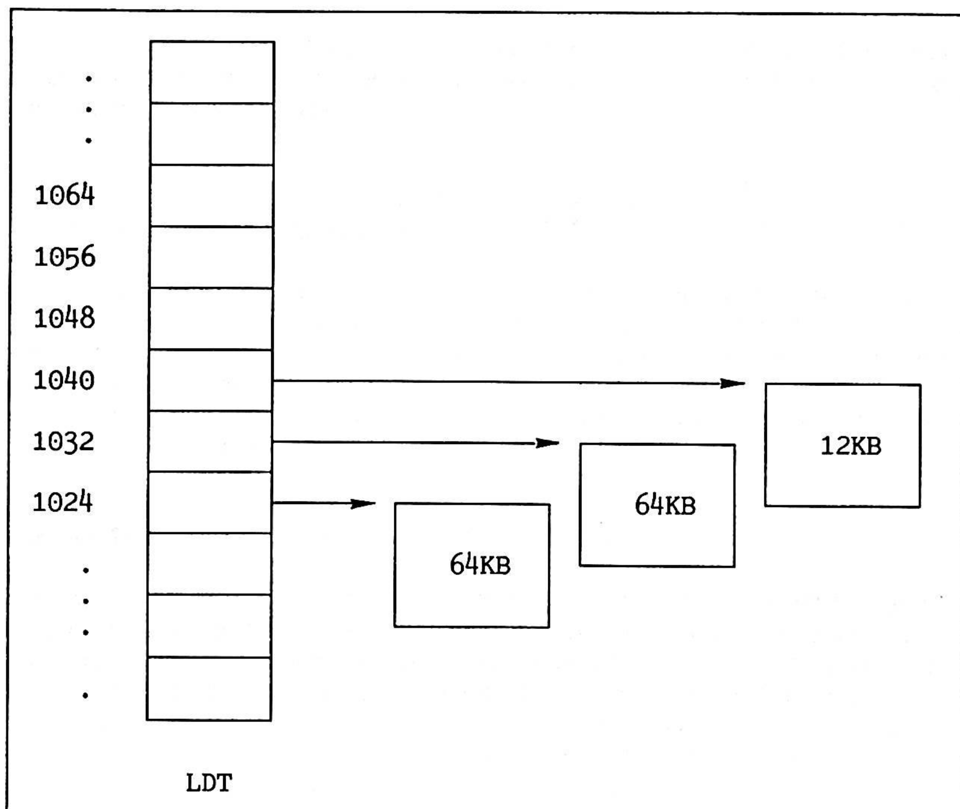
אם דרוש שטח זיכרון הגדול מ-64KB, צריך לשקול שימוש במאפיין זה של מישק התכנות.

DosAllocHuge מקצה גוש בזיכרון הגדול מהגודל המקסימלי של הסגמנט (64KB). התהליך מודיע למערכת כמה סגמנטים בני 64KB הוא רוצה, ומהו הגודל של הסגמנט האחרון (עד 64KB). התהליך מודיע למערכת גם את הגודל המקסימלי שאליו יגיע גוש הזיכרון הזה אי-פעם. התהליך משתמש ב-DosGetHugeShift כדי לחשב את ערכי הבורר המתאימים לסגמנטים הנוספים בגוש הזיכרון הגדול.

הבט על הדוגמה בתרשים 7. התהליך מבקש שטח זיכרון בן 140KB. מוחזר לו ערך בורר "1024", המצביע על סגמנט בן 64KB. על-ידי שימוש ב-DosGetHugeShift, התהליך מגלה שהוא צריך להוסיף 8 לערך הבורר הראשון (1024) כדי לקבל את הערך של הבורר (1032) שמצביע על ה-64KB השני בשטח בן 140KB בזיכרון. על-ידי הוספת 8 ל-1032, יחושב ערך הבורר השלישי. ערך הבורר השלישי מצביע על סגמנט המכיל 12KB. שלושת הסגמנטים מסתכמים ב-140KB זיכרון.

סגמנטים שהוקצו בדרך זו מתירים DosGiveSeg, DosGetSeg וביטול סגמנט. ההבדל היחיד בין זה לבין זיכרון שהוקצה עם DosAllocSeg הוא בכך, שהקריאות המתאימות משתמשות רק בערך הבורר הראשון עבור כל גוש הנתונים הגדול הזה.

DosGetHugeShift משיגה את הערך שצריך להתוסף לערך הבורר הראשון המוחזר כתוצאה מקריאה ל-DosAllocHuge. על-ידי הוספה של הערך הזה לערך הבורר הראשון נקבע ערך הבורר השני, שבו ישתמש התהליך כדי להכנס לגוש 64KB השני בזיכרון. כל אחד מערכי הבורר הבאים לאחר מכן מחושבים בדרך דומה.



תרשים 7. הקצאת זיכרון "גדול" של 140KB

DosReallocHuge משנה את הגודל של גוש הזיכרון שהוקצה במקור על-ידי **DosAllocHuge**. ניתן להגדיל רק עד למקסימום כפי שנקבע ב-DosAllocHuge המקורי.

הקצאות משנה בזיכרון

OS/2 מספקת קבוצה של מישקי תכנות שהתהליך יכול להשתמש בהם כדי להקצות יחידות זיכרון קטנות לתוך סגמנט יחיד. מאפיין זה חשוב, מאחר ואין זה יעיל ליצור סגמנט חדש לכל יחידה קטנה של זיכרון שהתהליך זקוק לו.

DosSubSet מודיעה למערכת להכין סגמנט לשימוש התוכנה להקצאות משנה (memory suballocation package). חייבים להשתמש במישק זה, אם גודל הסגמנט הוגדל עם DosReallocHSeg. החבילה להקצאות משנה משתמשת בסגמנטים שנוצרו על-ידי DosAllocSeg ו-DosAllocShrSeg.

DosSubAlloc מקצה בלוק של זיכרון בתוך הסגמנט שנוצר ואותחל על-ידי DosSubSet. התהליך מעביר את גודל הבלוק הרצוי ואת ערך הבורר של הסגמנט למישק התכנות. התהליך מקבל כתשובה את הכתובת היחסית של הבלוק בתוך הסגמנט. הבלוק המקסימלי שניתן להקצות יכול להגיע עד 8 בתים פחות מגודל הסגמנט. כל הבלוקים ניתנית להקצאה בכפולות של 4 בתים.

DosSubFree משחררת בלוק של זיכרון שהוקצה על-ידי DosSubAlloc. ערך הבורר של הסגמנט, גודל הבלוק, והכתובת היחסית של הבלוק בסגמנט מועברים למישק התכנות. אם המידע אינו עקבי עם הסידור של הבלוק בתוך הסגמנט, המערכת מודיעה על שגיאה.

ניהול זיכרון בהתקנים חיצוניים

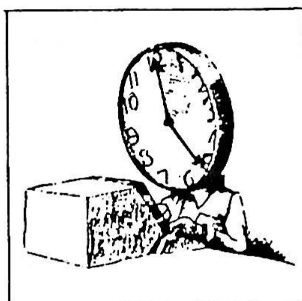
במבוא דנו בהתקני ק/פ ממופי זיכרון ובהתקנים עם גישה ישירה לזיכרון (DMA). מישק התכנות המאפשר את ניהול הזיכרון אינו מכיל מספיק פונקציות התומכות בהתקנים אלה. הפונקציות שכיסינו אינן מאפשרות ליישום לבקש להכנס לכתובת פיסית מסוימת. בעוד שיישום יכול ליצור סגמנטים בזיכרון ולהכנס אליהם, אין הוא יכול להודיע למערכת למקם את הזיכרון הזה או להכנס לכתובת פיסית מסוימת בזיכרון המערכת.

התקני ק/פ ממופי זיכרון

כדי להשתמש בהתקני ק/פ ממופי זיכרון, צריכה להיות ליישום האפשרות להכנס לכתובת פיסית מסוימת בזיכרון הפיסי המוקדש לק/פ ממופה זיכרון. פרק 6, אשר דן בניהול התקנים מבוקרי-פסקים עוסק בדרך שבה device driver יכול ליצור ערך בורר שהיישום משתמש בו כדי להכנס לזיכרון הפיסי בהתקן הק/פ.

התקנים עם גישה ישירה לזיכרון

התקנים המשתמשים בגישה ישירה לזיכרון (DMA) כדי להעביר נתונים לזיכרון המערכת וממנו צריכים תמיד את הכתובת הפיסית בזיכרון המערכת. אבל כפי שהסברנו כבר, היישום אינו מטפל בכתובת הפיסית של הסגמנט שבו הוא משתמש לאגירה של נתונים. פרק 6 מסביר איך ה-device driver משיג אפשרות מיעון לזיכרון פיסי זה.



פרק 3

ריבוי משימות ויישומים

בפרק הקודם חקרנו איך OS/2 מספקת מרחב כתובות המספיק כמעט לכל יישום, או קבוצת יישומים העולה על הדעת. שיפור נוסף ב-OS/2 לעומת ה-DOS הוא האפשרות שיש ליישומים לבצע מספר משימות בו-זמנית והאפשרות שיש למשתמש לעבוד עם יותר מיישום אחד בו-זמנית.

בעולם ה-DOS אפשר להפעיל במחשב רק יישום אחד בו-זמנית. זאת אומרת שהמשתמש חייב להשלים את כל העיבוד הדרוש לתכנית A לפני שהוא יוכל להריץ את תכנית B. מדוע חשוב להריץ יותר מיישום אחד בו-זמנית?

אם היינו עושים את עבודות הבית לפי הכללים של פעילות אחת בלבד בו-זמנית, היום שלנו היה עובר בצורה מאוד לא יעילה. היינו מעמיסים את כל כלי המטבח לתוך מדיח הכלים, מפעילים אותו ולא עושים דבר עד שהוא מסיים. רק אז היינו מרוקנים ממנו את הכלים ומסדרים אותם. לאחר מכן היינו מכניסים את כל הבגדים המלוכלכים לתוך מכונת הכביסה ומחכים שהיא תסיים את פעולתה, מעמיסים את מייבש הכביסה ומפעילים אותו. אם נשארו עוד בגדים שיש צורך לכבס אותם, לא היינו יכולים לעשות דבר עד אשר מייבש הכביסה יסיים את פעולתו. רק אז היינו יכולים לכבס בגדים נוספים. כמובן שנוהל עבודה זה אינו מאפשר ניצול נכון של הזמן שלנו. אם היינו יכולים להפעיל את מדיח-הכלים, מכונת-הכביסה ומייבש הכביסה בו-זמנית, הזמן הכולל שהיה עובר היה קטן בהרבה, והיינו יכולים לבצע עבודות נוספות שהיה צורך לעשותן.

דבר זה נכון גם לגבי משתמשים בתחנת עבודה. משתמש ירצה להריץ מספר תכניות, אשר כמעט אינן דורשות הידברות, אך זמן הביצוע שלהם הוא ארוך. להלן מספר דוגמאות של תכניות עם המאפיינים האלה:

- חישוב של גיליון אלקטרוני גדול.
- הידור של תכנית.
- קבלת נתונים בקווי תקשורת.
- הדפסת קובץ.

בזמן שכל הפעילויות האלו מתבצעות, המשתמש יכול לפעול עם תכנית שדורשת מידה רבה של הידברות, כמו עורך (editor) או מעבד תמלילים. על-ידי ביצוע כל הפעילויות האלו במקביל, המשתמש יכול להעלות בצורה משמעותית את פריון העבודה שלו. משך הזמן שדרוש להריץ מספר תכניות במקביל אינו שווה לסך כל הזמנים, אילו היינו מריצים כל יישום בנפרד. השוויון אינו קיים, מאחר ורוב היישומים אינם צריכים את המעבד ב-100% של הזמן. בד"כ היישום צריך לחכות לנתונים שייקראו מהתקן ק/פ, או לחכות שייכתבו לשם. בזמן שהיישום מחכה שפעילות זאת תתבצע, המעבד יוכל לבצע עבודה מועילה אחרת. התמיכה של OS/2 בסביבה רבת משימות ומרובת יישומים מבוססת על ניצול מקסימלי של המעבד.

תחת DOS, היכולת של המעבד לבצע עבודה ביעילות מוגבלת, מכיון שסביבת DOS מאלצת אותו "לחכות" לסיום פעולות הק/פ לפני שהוא יכול לבצע את העבודה הבאה. המעבד אינו נכנס למעשה למצב של "המתנה". בפועל, הוא ממשיך לשאול אם פעולת הק/פ הסתיימה עד שהוא מקבל תשובה חיובית. סוג הפעולה הזאת ידוע בשם **לולאה סיבובית** (spin loop).

לפעמים הפעלה של מספר עותקים של אותו יישום יכולה להיות מאוד מועילה. משתמשים רוצים למשל, שמספר עותקים של עורך הטקסט יהיו פעילים כדי שיהיה ניתן לעבוד בו-זמנית על קבצים שונים של רשימות שונות בנושאים שונים. על-ידי שימוש במספר עותקים של אותו יישום בו-זמנית, המשתמש אינו צריך לבצע את התהליך של שמירת קובץ אחד והבאה של אחר. חשוב מאוד שהשיטה המאפשרת מעבר בין יישום אחד לשני תהיה מהירה וקלה.

אנו יכולים להבין בקלות את היתרון שנשיג אם נוכל להריץ יותר מיישום אחד בו-זמנית. אבל מה לגבי הרצה של מספר משימות בו-זמנית? OS/2 משתמשת בתמיכה שלה במספר משימות בו-זמנית כדי לאפשר את הקיום של סביבה מרובת יישומים. הצורך בהפעלה של מספר משימות בו-זמנית קיים אפילו לגבי יישום אחד.

תמיכה במספר משימות בו-זמנית מאפשרת ליישום לעשות יותר מאשר פעולה אחת בעת ובעונה אחת. מאחר ויש מעבד אחד בלבד במערכת, היישום אינו מבצע בפועל יותר מדבר אחד בו-זמנית, אולם המערכת משתמשת במעבד בצורה כזו, שנראה לנו שהיא מבצעת פעולות אחדות באותו זמן.

במערכת DOS, יישום יכול לעשות רק משימה אחת בו-זמנית. אפשר לחשוב על משימה כנתח של עבודה שצריך להתבצע על-ידי המעבד. יישומים חייבים לבצע את המשימות שלהם בצורה **מסונכרנת** (synchronously), בדרך שבה כל משימה חייבת להתבצע לאחר משימה אחרת. לדוגמה, לאחר בקשה לקרוא קובץ מהדיסק, אין יישום ב-DOS יכול לבצע דבר אחר עד אשר הקריאה הושלמה והנתונים נקראו מהדיסק לתוך הזיכרון. יישום ב-DOS יכול לשאול את המערכת האם יש תווים שהוא צריך לקרוא מהמקלדת. פעולה זו נקראת **חקירה** (polling). הסיבה לחקירה זו היא שהיישום אינו רוצה להגיע למצב שבו הוא יתקע בהמתנה להקשה מהמקלדת.

בסביבה רבת משימות, יישום אינו צריך לבצע את המשימות בזו אחר זו, או בצורה מסונכרנת. ליישום יכולה להיות יותר ממשימה אחת פעילה בו-זמנית. למשל, הוא יכול להיות במצב שבו משימה אחת תהיה קריאה מהמקלדת ומשימה אחרת תהיה קריאת נתונים מהדיסק. בזמן ההמתנה לקריאת הנתונים מהדיסק, היישום יכול להשתמש במעבד כדי לקבל הקשות מהמקלדת ולעשות כל דבר אחר שדרוש. מאחר ואין תזמון בין המשימות בכל נקודה ונקודה, נוכל להגיד שהמשימות מתבצעות בצורה **אסינכרונית** (asynchronously).

בגלל האופי המורכב של הסביבה רבת המשימות, המערכת מספקת שירותים רבים על-מנת להבטיח שיישום או קבוצת יישומים ייכתבו בצורה נכונה. לדוגמה, המשימות ביישום חייבות לבצע את הדברים הבאים:

- להתחיל, לעצור ולפקח על משימות.
- להתקשר זו עם זו.
- לסנכרן פעילויות.
- להשתמש באופן סידרתי במשאבי המערכת.

ללא אפשרויות אלו, לא היה אפשר לבצע עבודה בצורה יעילה בסביבה רבת משימות, מאחר וזה היה גורם לאי-סדר נוראי. בפרק זה אנו חוקרים את הדרך שבה המישק לתכנות יישומים ב-OS/2 תומך בסביבה רבת משימות.

בסביבת OS/2 אין משתמשים במונח "משימה" (task). במקומו משתמשים בשלושת המונחים הבאים כדי לתאר את היישויות המתייחסות לסביבה רבת המשימות ומרובת היישומים של OS/2:

- session (מהלך עבודה).
- process (תהליך).
- thread (יחידת ביצוע - "חוט").

אנו מתארים את האפשרויות הסביבה רבת המשימות ב-OS/2 מנקודת המבט של session, process ו-thread.

סביבה מרובת יישומים

מנקודת המבט של המשתמש, יישום כולל בד"כ תכנית הקוראת מהמקלדת או מעכבר, מציגה נתונים על המסך, ומבצעת עיבוד נתונים. בטרמינולוגיה של OS/2 לפנינו **session** (מהלך עבודה).

היישום שכרגע מקבל הקשות מהמקלדת ומציג נתונים על המסך נקרא **foreground session** (מהלך חזית). שאר התכניות במערכת ה-OS/2 נקראות **background sessions** (מהלך רקע), והן מקבלות זמן מעבד וממשיכות להתבצע. השוני היחיד בין שתי קבוצות התכניות הוא בכך, שהמשתמש יכול להתקשר רק ל-session אחד בו-זמנית: לזה הנמצא ב-foreground, כלומר - בחזית.

אין זה הגיוני לאפשר לכל תכנית שרצה במערכת לכתוב על המסך בו-זמנית. אין זה גם הגיוני שכל תכנית במערכת תקבל כל תו שהמשתמש מקליד מאחר וזה היה מבלבל את המערכת. כאשר background session מנסה לכתוב נתונים למסך, הוא מופנה לחלק מסוים בזיכרון המערכת שהוא ייצוג לוגי של המסך. הנתונים אינם מוצגים למעשה על המסך, אלא בשטח בזיכרון הנקרא **מאגר תצוגה לוגי** (logical video buffer). כאשר background session עובר להיות foreground, מאגר התצוגה הלוגי ששייך לו עובר למסך. תמונת המסך הקודמת נשמרת במאגר התצוגה הלוגי של ה-session הקודם. באותו אופן, כאשר המשתמש מקיש במקלדת או מזיז את העכבר, הנתונים עוברים ל-session הנמצא ב-foreground בלבד. התכניות הפועלות ב-background sessions מניחות שהמקלדת ו/או העכבר אינם בשימוש.

סביבת ה-DOS מקבלת זמן מעבד רק כאשר היא הופכת ל-foreground session. כאשר סביבת ה-DOS עוברת להיות background session, היא אינה מקבלת זמן מעבד והיא מוקפאת עד אשר היא מוחזרת ל-foreground. זאת מאחר ויישומים בסביבת הביצוע של DOS אינם "מתנהגים כראוי". לדוגמה, יישום DOS הנמצא ב-background session יוכל לכתוב ישירות למסך עצמו, כלומר לכתוב על המסך השייך ל-foreground session, ולא כיישומי OS/2 אשר כותבים למאגר תצוגה זמני.

SESSIONS רבים - נקודת המבט של המשתמש

איך מתורגמת הקונספציה של ה-session לפעולות שבהן המשתמש יכול לשלוט? **בורר התכניות** (program selector) **ומנהל ה-sessions** מבצעים משימה זו.

בורר התכניות מספק תפריט על מסך מלא המאפשר למשתמש לעשות את הדברים הבאים:

- לבחור תכנית להרצה מתוך רשימה של תכניות.
- להתחיל את מעבד הפקודות (command processor).
- להעביר את המסך לתכנית שכבר הופעלה.

הבחירה של המשתמש הופכת להיות foreground session. המשתמש יכול להוסיף תכניות חדשות לרשימת התכניות הניתנות להרצה, או לבטל ממנה תכנית כלשהי.

אם סביבת ה-DOS מסופקת כחלק מהתצורה של המערכת, המשתמש יכול לבחור את הסביבה הזאת מתוך הרשימה של התכניות שנמצאות כבר בהרצה. אם המשתמש בוחר בסביבת ה-DOS, המסך מתחלף ומופיע המנחה של ה-DOS ומעבד הפקודות של ה-DOS (command.com) עובר להיות פעיל. המשתמש יכול לבחור את מנחה (prompt) ה-OS/2 מתוך רשימת התכניות הניתנות להרצה. לאחר שנבחר מנחה ה-OS/2, הוא מוצג על המסך ומעבד הפקודות (cmd.exe) של ה-OS/2 עובר להיות פעיל לגבי ה-foreground session הנוכחי. מרגע שמנחה OS/2 מופעל, ניתן להריץ תכניות OS/2, בדיוק כמו שניתן להריץ יישום DOS ממנחה ה-DOS. המשתמש יכול ליצור קובץ מיוחד הנקרא **קובץ אצווה** (batch file) המכיל סידרה של פקודות שמוכנסות באופן נורמלי באמצעות מנחה הפקודות. הדבר מאפשר לקבוע מראש סידרה מורכבת של פקודות שתשמש מפעיל פחות מתוחכם של תחנת העבודה.

מנהל ה-session מספק צירוף של מקשים שהמשתמש יוכל בעזרתם לעבור ל-session הבא (תכנית הנמצאת במצב ריצה). צירוף מקשים אחר מאפשר למשתמש להעלות מיד את התפריט של בורר התכניות. בעזרת אפשרות זו, המשתמש יכול לשנות בקלות את התכנית שאיתה הוא רוצה לעבוד. למשתמש גם קל מאוד להריץ תכניות חדשות, או לבחור איזה session יהפוך ל-foreground session.

הכוח של מנהל ה-sessions היה מוגבל אילו היה אפשר להפעיל אותו רק בצורה הידברותית. החלק הבא מתאר את מישקי התכנות של מנהל ה-session. המישק מאפשר לתכנית הרצה ב-session אחד להתחיל ולשלוט ב-session אחר. אפשר להשתמש במאפיין חזק זה של OS/2 בצורה אוטומטית כדי להפעיל sessions חדשים ולקבוע באמצעותו איזה מהם יופיע למשתמש כ-foreground session.

מישקי התכנות ל-SESSION

DosStartSession יוצרת ומתחילה תכנית חדשה ב-session נפרד. התכנית החדשה מתקשרת עם המשתמש ללא תלות בתכנית שהתחילה אותה. המבקש מגדיר את הכותרת של התכנית, הכוללת את השם המלא של התכנית ואת הפרמטרים שיש צורך להעביר לתכנית החדשה. הכותרת של התכנית מופיעה בתפריט של התכניות הנמצאות כבר בביצוע, שהוא חלק מבורר התכניות. שם התכנית מכיל את אות

הכונן המתאים ואת המסלול (path directory) שדרכו ניתן להגיע לתכנית, ואת השם של הקובץ המכיל את התכנית. הפרמטרים הינם חלק מהקלט המועבר לתכנית כאשר מתחילים אותה בסביבת הביצוע של OS/2. הם מגדירים לתכנית איך לפעול כאשר היא נקראת.

בנוסף לכך, אפשר להשתמש במישק זה כדי להעלות את מעבד הפקודות של OS/2 כתכנית (cmd.exe) שיש צורך להתחיל אותה ב-session אחר. משתני הקלט הם היישום שממנו יש צורך להפעיל את מעבד הפקודות של OS/2. מעבד הפקודות מכוון לסיים את ה-session כאשר היישום סיים את עבודתו, או לאפשר ל-session לרוץ בהנחייה של מעבד הפקודות כאשר היישום יגמור את עבודתו. אם בוחרים את היישום מתוך רשימת התכניות הניתנות להרצה, התכנית תתחיל לרוץ ללא מעבד הפקודות.

אם התכנית המבקשת שירות DosStartSession נמצאת ב-foreground session, היא יכולה לבקש שהתכנית שהיא עומדת להריץ תעבור להיות foreground session. אם לא כן, התכנית שהותחלה חייבת להיות background session חדש.

התכנית החדשה ניתנת להרצה כיישות בלתי תלויה לחלוטין מהתכנית שהתחילה אותה. אולם התכנית המבקשת יכולה לבקש שיתקיימו יחסים של אב/בן בין ה-session של המבקש לבין ה-session של התכנית החדשה. ה-session של המבקש הופך להיות session האב, והתכנית החדשה הופכת להיות session הבן. אין לבלבל את יחסי אב/בן אלה עם יחסי אב/בן הקיימים בין תהליכים שנדון בהם במישק התכנות DosExecPgm. על-ידי הפיכת ה-session החדש לילד שלו, יכול session האב לשלוט ב-session בעזרת מישקי תכנות אחרים של מנהל ה-session. session האב יכול לשלוט רק בצאצאים ישירים שלו. הוא אינו יכול לשלוט בצאצאים של הבן (כלומר כל session שהותחל על-ידי הבן). אם session האב מתבטל, הבנים שלו מתבטלים גם הם.

אם התכנית המבקשת יוצרת יחסי אב/בן, מספר הזיהוי של session הבן חוזר לאב. מספר זיהוי זה משמש בקריאות למישקי תכנות אחרים של מנהל ה-session. בנוסף לכך, התכנית המבקשת יכולה לקבוע שטח בזיכרון כדי לקבוע מתי מסתיים session הבן ומה היה הקוד שהוחזר ממנו. מנגנון התורים ב-OS/2 מבצע זאת. אנו נדון בפונקציית התורים ב-OS/2 מאוחר יותר בפרק זה.

DosSelectSession session האב הופך את אחד מילדיו ל-foreground session. זה יכול לקרות רק אם session האב או אחד מהצאצאים שלו נמצאים כרגע ב-foreground session. האב משתמש במספר הזיהוי של ה-session כדי לזהות את הבן שהוא רוצה להפוך ל-foreground session. בעזרת מישק זה, ה-session שהמשתמש עובד איתו כרגע בצורה הידברותית יוכל להיות מבוקר בצורה אוטומטית.

התכנית הקוראת יכולה גם לבקש שה-session שלה יהפוך ל-foreground session. פעולה זו יכולה להתבצע רק אם אחד מהצאצאים שלה נמצא כרגע ב-foreground session.

DosSetSession session האב יכול לקבוע מספר מאפיינים ל-session הבן. האב משתמש במספר הזיהוי של הבן כדי לזהות את session הבן.

session האב יכול לקבוע אם session הבן יהיה ניתן לבחירה, או שלא יהיה

ניתן לבחירה מתוך התפריט של בורר התכניות. ניתן עדיין לקבוע ש-session הבן יהיה foreground session בעזרת DosSelectSession.

session האב יכול לקשור אליו את אחד מבניו. זאת אומרת שאם המשתמש בוחר את session האב, אז session הבן יהפוך בצורה אוטומטית ל-foreground session. אפשר לקשור רק בן אחד בו-זמנית ל-session האב. הקשירה משפיעה על בחירה הנעשית על-ידי המשתמש, אך אינה משפיעה על בחירה הנעשית דרך DosSelectSession. session האב יכול גם לשחרר את session הבן.

DosStopSession session האב יכול להפסיק את פעולתו של בן מסוים על-ידי שימוש במספר המזהה של אותו session, או הוא יכול להפסיק את פעולתם של כל צאצאיו. אפילו אם הבקשה לפונקציה זו הושלמה, אין המבקש יכול להניח שהבן הפסיק את עבודתו, עד אשר הוא קיבל הודעה על כך ממנגנון התורים שהופעל כאשר session הותחל בעזרת DosStartSession.

תהליכים ו-THREADS

אחד מהתפקידים של OS/2 הוא לנהל את השימוש במשאבי המערכת. אנו יודעים כרגע שהגישה למקלדת ולמסך מנוהלת כ-session. המסך הפיסי והמקלדת כפופים למרותו של ה-foreground session.

ישנם סוגים נוספים של משאבים במערכת. כניסה לסגמנט מסוים בזיכרון או לקובץ (על-ידי שימוש ב-file handle) הן דוגמאות למשאבי מערכת. לכל תהליך יש טבלת מתארים מקומית (LDT) משלו המאפשרת לו להכנס לזיכרון שבבעלותו. לכל תהליך יש גם מערכת של קבצים, צינורות, תורים ואתתים הקשורה אליו. היישות שבבעלותה נמצאים משאבי המערכת ב-OS/2 נקראת **תהליך** (process).

יישום מתחיל כתהליך בתוך session. לדוגמה, מעבד הפקודות מאפשר למשתמש להתחיל יישום משורת הפקודה של ה-session. הסיבה מדוע יישום אינה מלה נרדפת לתהליך היא בכך, שהתהליך המקורי היוצר את היישום יכול להתחיל תהליכים נוספים, שגם הם חלק מהיישום. כל התהליכים של היישום יכולים להתחלק (להיות שותפים) במקלדת ובמסך מאחר והם נמצאים באותו session. אולם הם אינם יכולים להתחלק באותו זיכרון (פרט למקרה שבו מאפשרים זאת על-ידי הגדרת סגמנטים משותפים) ואין הם יכולים להתחלק בגישה לאותם קבצים. הפרדת הבעלות על משאבי המערכת מראה שקיימת מחיצה מוגדרת היטב בין תהליכים.

מחיצה זו יכולה להיות מאוד מועילה ליישומים גדולים ומורכבים. המחיצה, שהיא למעשה אנלוגיה על הקונספציה של הבעלות על משאבי המערכת, מאפשרת לתכנן את היישום, כך שיהיה ניתן לשבור אותו לקטעים קטנים שיכולים לפעול זה עם זה רק בדרכים מוגדרות היטב.

תהליכים שונים מספקים מנגנון השומר על השלמות של חלקים שונים של היישום. תהליך אחד אינו מקבל באופן אוטומטי את האפשרות לשנות את התוכן של קבצים או נתונים של תהליך אחר. ניתן להשתמש בזה להגנה על מידע רגיש מפני חלקים שונים של אותו יישום. מערכת ההפעלה עורכת מעקב אחר משאבי המערכת שבבעלות התהליך ועוזרת לאושש את משאבי המערכת אם התהליך לא הסתיים בצורה מוצלחת.

התהליך ב-OS/2 מאפשר לשבור יישומים גדולים ומורכבים לקטעים קטנים שיכולים להשפיע זה על זה בדרכים מוגדרות היטב. על-ידי תכנון היישום כתהליכים נפרדים, אנו יכולים להקטין את המורכבות הכוללת שלו. קבוצות שונות או אנשים שונים, יכולים להתקין את התהליכים השונים עם תכנון ברור מאוד כיצד הם יפעלו זה עם זה. ארכיטקטורת התהליכים ב-OS/2 מאפשרת את התכנון הנדרש.

המחיצה בין התהליכים השונים דורשת מנגנונים מוגדרים היטב שיאפשרו לתהליכים לפעול זה עם זה. הפעולה ההדדית הזו נקראת **תקשורת בין תהליכים** (interprocess communications) והיא מכוסה בפרק זה. אנו יודעים כבר איך תהליכים שונים יכולים להכנס לאותו סגמנט בזיכרון. תהליכים הפועלים זה על זה דרך מנגנונים אלה אינם בהכרח חייבים להיות חלק מאותו יישום.

בבעלות. תהליך נמצאות גם יחידות ביצוע. יחידת ביצוע נקראת **thread**. אפשר לדמיין זאת כסידרה של פקודות המבוצעות בזו אחר זו. אנו יכולים להשוות יישום בסביבת DOS ל-thread, מאחר וכל התכנית מתבצעת בצורה מסונכרנת מהתחלה עד הסוף. בסביבת DOS לא יכול לקרות ששני מערכים של פקודות יבוצעו בו-זמנית או בצורה אסינכרונית. ב-OS/2, כל המערכים של הפקודות המתבצעים בצורה בלתי-תלויה זה מזה (בצורה אסינכרונית) נקראים threads.

תהליך מורכב מ-thread אחד לפחות אבל הוא יכול להיות מבוסס גם על threads רבים. כל ה-threads של התהליך מתחלקים בצורה שווה במשאבי המערכת הנמצאים בבעלות התהליך. לדוגמה, לשני threads של אותו תהליך יש אפשרות גישה לאותם סגמנטים בזיכרון, ואין שום דרך לאחד לבטל את הזכות של חברו. כאשר תהליך מורכב מ-threads רבים, הם נחשבים ל"מחוברים בצורה הדוקה" כי הם חייבים לשתף פעולה זה עם זה. אם הם לא ישתפו פעולה, הם עלולים לשנות זה את זה, או לשנות את משאב המערכת ששניהם פונים אליו מבלי להיות מודעים למעשה זה.

כאשר שני threads מתבצעים בצורה אסינכרונית זה לזה, זאת אומרת שכל אחד מהם אינו יודע מה השני עושה. לדוגמה, thread אחד של התהליך יכול לקרוא נתונים מהמקלדת ולכתוב אותם על המסך, ו-thread אחר של אותו תהליך יכול לקרוא מקובץ ולכתוב על המסך. היכולת להריץ מספר threads במקביל היא בעלת ערך רב, כי היא מאפשרת לתהליך להמשיך לעשות עבודה מועילה, אפילו אם צריך לחכות שחלק אחר במערכת ישלים ביצוע בקשה שלו. אפילו אם פעולות הק/פ בקובץ נמשכות זמן רב, המשתמש יוכל עדיין לראות את היישום מגיב לקלט מהמקלדת. הסיבוך היחיד בדוגמה זו הוא בכך ששני ה-threads מעדכנים את המסך. אם כל אחד מהם היה פועל על חלק אחר מהמסך לא היתה קיימת בעיה. אם שניהם מעדכנים את אותו חלק של המסך, שניהם חייבים בנקודות מסוימות ומוגדרות היטב במהלך הביצוע לסנכרן את הפעולות שלהם זה עם זה.

למרות שתהליך הוא יישות ב-OS/2 שבבעלותה נמצאים משאבי מערכת, המערכת צריכה עדיין לערוך מעקב אחר דברים מסוימים ביחס ל-thread:

- מספר הזיהוי של ה-thread ID) thread). בזמן שנוצר כל thread של תהליך, הוא מקבל מספר זיהוי ייחודי. מספר זיהוי זה משמש threads אחרים בתהליך כדי לזהות thread מסוים בתהליך.
- מחסנית (stack). כל thread בתהליך דורש שטח נפרד לאוגרים שלו.
- אוגרים במעבד (אם אינם מבצעים דבר כרגע). לכל thread בתהליך יש מערך נפרד של אוגרים במעבד שחייבים לעקוב אחריו.

- מצב ניתוב. מתזמן העבודות (scheduler) ב-OS/2 צריך לדעת האם ה-thread ניתן לביצוע, או אם הוא מחכה כרגע למשהו שיקרה במערכת לפני שיהיה ניתן לבצע אותו.
- עדיפות (ידון בהמשך). לכל thread יש עדיפות ביצוע ייחודית הקשורה אליו.

OS/2 דורשת את הפרטים האלה כדי שתוכל לנהל יחידות ביצוע שונות (threads).

פחות יקר לתהליך, במונחים של זמן, להתחיל thread חדש מאשר להתחיל תהליך חדש. thread הוא רק יחידת ביצוע חדשה בתוך אותה סביבה של התהליך. באופן בסיסי ה-thread החדש יוצר סביבת עבודה קטנה. לעומת זאת, תהליך חדש יוצר סביבת תהליך חדשה, הכוללת טבלת מתארים מקומית חדשה (LDT) ומערכת חדשה של מקשרים לקבצים.

יישום המתבסס על תהליך אחד עם threads רבים, הוא מבנה של יחידות ביצוע רבות שכולן מתבצעות בצורה אסינכרונית. כל יחידות הביצוע הנפרדות מתחלקות באופן שווה באותם משאבי מערכת, כמו גישה לסגמנטים ומקשרים לקבצים. המבנה של יישום המתבסס על תהליכים רבים זהה לדוגמה הקודמת, פרט לכך שליחידות הביצוע השונות יש מערכים נפרדים של משאבי מערכת. לכן הן אינן יכולות להכנס זו למשאבי זו (לדוגמה, סגמנטים או קבצים) אם לא ניתן לכך אישור במפורש.

בפרק 4 תמצא שתי דוגמאות תכנות המתארות תהליכים ו-threads. הדוגמה הראשונה מראה תבנית פשוטה של thread שמשתמש באתת (semaphore) של ה-RAM ובמספר מישקי תכנות כדי להודיע על פסק ולסנכרן את הביצוע של ה-threads. הדוגמה השניה מפתחת את תבנית ה-thread הזו על-ידי הצגה של תהליך נפרד. בדוגמה זו, מנגנון הצינור (pipe) לתקשורת בין תהליכים, באמצעות העברת נתונים בין תהליכים. אנו נתאר את מישקי התכנות השונים המתייחסים לתהליכים ול-threads מאוחר יותר בפרק זה.

מישקי תכנות לתהליכים

תכנית ב-OS/2 מותחלת כתהליך חדש. כאשר תכנית חדשה מתחילה, מועבר אליה **סגמנט סביבה** (enviroment segment), אשר מורה לה מה היא צריכה לעשות. סגמנט הסביבה דומה לקידומת סגמנט התכנית (Program Segment Prefix - PSP) שמועברת לתכנית DOS בזמן התחלתה.

התהליך משתמש ב-DosExecPgm כדי ליצור ולהתחיל תהליך חדש. לאחר שנוצר תהליך חדש, מועברים אליו פרמטרים המציינים מצבי פעולה (argument strings) ופרמטרים המציינים את הסביבה (environment strings) כחלק מסגמנט הסביבה. התהליך החדש קובע מה יהיה המבנה של סגמנט הסביבה והתהליך היוצר קובע את התוכן של סגמנט הסביבה. מעבד הפקודות ב-OS/2 תמיד יעביר את שם התכנית ואת כל מה שהוכנס על-ידי המשתמש בשורת הפקודה כמחרוזות של ארגומנטים הקובעים מצב פעולה. המשתמש יכול לקבוע באמצעות משתני הסביבה בפקודת SET את הארגומנטים של הסביבה. תכנית יישום יכולה לקבוע תו מוסכם, שאם המשתמש ישתמש בו, התכנית תתנהג בצורה מסוימת. OS/2 תומכת במשתני סביבה מסוימים שניתנים לשינוי על-ידי המשתמש. לדוגמה, מחרוזת תווי הסביבה PATH קובעת את מסלולי החיפוש של קבצים הניתנים לביצוע.

כאשר תהליך אחד יוצר תהליך אחר, התהליך היוצר נקרא תהליך האב והתהליך שנוצר נקרא תהליך הבן. אין לערב יחסי אב/בן בין תהליכים עם יחסי אב/בן בין sessions.

תהליך הבן **יורש** (inherits) מאפיינים מסוימים של תהליך האב. לדוגמה:

- מקשרים (handles) ל:
  - קבצים.
  - צינורות.
  - התקנים סטנדרטיים (ק/פ).
  - התקני תווים.
- סגמנט הסביבה.
- עדיפות ביצוע.
- session.

תהליך הבן יורש את מקשרי הק/פ הסטנדרטיים לקבצים. אם נוסיף לכך שהאב יכול לשלוט להיכן יצביעו מקשרי הקבצים, נקבל שתהליך האב יכול לשלוט להיכן תהליך הבן ישלח את הקלט והפלט שלו (בהנחה שהוא משתמש במקשרי הקבצים הנכונים). המערכת משתמשת במשמעות הזו של "ירושה" כדי להסדיר את **הניתוב** (redirection). לדוגמה, המשתמש יכול להתחיל יישום באמצעות מעבד הפקודות שמקבל את הקלט שלו מקובץ מסוים, ואז לגרום לכך שפלט של היישום יעבור לקובץ מסוים אחר או ליישום אחר. תהליך יכול לשלוט על תהליך אחר על-ידי יצירה שלו ושל סגמנט הסביבה שלו, להפסיק את פעולתו, לשנות את עדיפות הביצוע שלו, לשלוט במקשרים שיעברו אליו ולעבוד איתו באמצעות המנגנונים לתקשורת בין תהליכים ב-OS/2.

תהליך יכול לקבוע שקבוצת פקודות מסוימת תתבצע, כאשר הוא יסיים את פעולתו. פקודות אלו נקראות שיגרת היציאה (exitlist routine). התהליך משתמש באפשרות זו כדי לקבוע איך יתנקו משאבי המערכת שלו לפני שהמערכת מסיימת את פעולתו. ספריית הקישור הדינמי משתמשת באפשרות זו כדי לנקות כל משאב שהיא מעמידה לרשות התהליך.

לתהליך יכולות להיות מספר שגרות יציאה, המתבצעות כאשר מפסיקים את פעולתו של היישום. ישנן הגבלות על מה ששיגרת יציאה יכולה לעשות. לדוגמה, שיגרת יציאה אינה יכולה להתחיל תכניות אחרות, והיא חייבת להסתיים כמה שיותר מהר.

DosExecPgm תכנית נוצרת ומותחלת כתכנית בן. תהליך האב מזהה את הקובץ המכיל את התכנית כשם קובץ מלא (מכיל את מספר הכונן ומסלול חיפוש במדריך), או רק כשם קובץ. אם שם הקובץ אינו מוגדר בצורה מלאה, יבוצע חיפוש לפי המשתנה הנוכחי המוגדר עבור PATH, עד אשר יימצא שם הקובץ המתאים המכיל את התכנית. שם התכנית מועבר גם הוא לתהליך הבן.

סביבה חדשה נוצרת עבור תהליך הבן. כפי שהסברנו קודם, תהליך הבן יורש מאפיינים מסוימים מתהליך האב שלו, אשר יכול לשלוט בחלק מהמאפיינים העוברים בירושה. אם תהליך האב רוצה לשלוט בסגמנט הסביבה שהבן יירש, הוא מעביר לתהליך הבן מחוון (pointer) לפרמטרים הקובעים מצב פעולה ומחוון לפרמטרים שיקבעו את הסביבה (צירוף של שניהם יוצר את סגמנט הסביבה).

תהליך האב שולט בצורה שבה יתבצע תהליך הבן ביחס לתהליך האב. **בביצוע מסונכרן**, ה-thread של תהליך האב לא ימשיך להתבצע מהנקודה שבה בוצעה הקריאה ל-DosExecPgm, עד אשר תהליך הבן השלים את עבודתו. קוד הסיום וקוד התוצאה של תהליך הבן מועברים אל תהליך האב. תהליך הבן מפרט את קוד התוצאה כאשר הוא מסיים את עבודתו. קוד הסיום מודיע לתהליך האב אם תהליך הבן הסתיים בצורה נורמלית, לא-נורמלית, או שעבודתו הופסקה על-ידי תהליך אחר לפני הסיום הנורמלי שלה.

בביצוע לא מסונכרן, ה-thread של תהליך האב ממשיך להתבצע בו-זמנית יחד עם תהליך הבן. כאשר הקריאה ל-DosExecPgm מסתיימת, מוחזר מספר הזיהוי של הבן לתהליך האב וה-thread של תהליך האב ממשיך להתבצע בזמן שתהליך הבן מתבצע. תהליך האב משתמש במספר הזיהוי של תהליך הבן עבור מישקי התכנות המאפשרים שליטה עליו. אם תהליך האב רוצה את קוד התוצאה של הבן, הוא משתמש במישק התכנות DosCwait, כדי לבקש שקוד זה יוחזר אליו.

DosCwait יחידת הביצוע, ה-thread של תהליך האב ממתינה לסיום של תהליך הבן. כאשר עובדים בצורה אסינכרונית, יש צורך להתחיל את תהליך הבן בעזרת DosExecPgm עם אופציה של החזרת קוד מצב ל-DosCwait.

ה-thread של תהליך האב יכול לבחור בין שני מצבי פעולה. במצב הראשון עליו לחכות עד אשר תהליך הבן יסתיים. בשני - הוא ימשיך להתבצע אם לא התקבלה הודעת סיום בזמן שבוצעה הקריאה ל-DosCwait. תהליך האב יכול לבחור לחכות לתהליך בן מסוים, או לכל אחד מהבנים. בנוסף לכך, הוא יכול גם לחכות לסיום של הצאצאים של תהליך הבן. קוד התוצאה וקוד סיום הפעולה של תהליך שהושלם מועברים לתהליך האב.

DosKillProcess תהליך יכול להפסיק (terminate) את פעולתו של תהליך הבן, או של תהליך הבן וכל הצאצאים שלו. DosKillProcess מפיק אות הנקרא SIGTERM ומעביר אותו לתהליכים, אשר אמורים להפסיק את פעולתם. תהליך יכול לקבוע שיגרה מיוחדת, אשר תטפל באות ה-SIGTERM ותשתלט על הביצוע, כאשר תהליך אחר מנסה להפסיק את פעולתו בעזרת DosKillProcess. אנו נדון בטיפול באותות מאוחר יותר בפרק זה.

DosExit מאפשר ל-thread להפסיק את פעולתו. ה-thread יכול לבקש שכל ה-threads בתהליך שלו יפסיקו את פעולתם, במטרה להפסיק את פעולת התהליך כולו. כאשר תהליך רוצה להפסיק להתבצע, הוא צריך לקרוא ל-DosExit ולבקש שהביצוע של כל ה-threads יופסק. הדבר יפסיק את פעולתו של כל thread שמערכת ההפעלה התחילה מטעמו של התהליך, אך הוא אינו יודע עליו. כאשר תהליך מפסיק את פעולתו, יתבצעו שגרות הסיום שנקבעו בעזרת DosExitList.

DosExitList מאפשר לתהליך לקבוע קבוצה של פקודות שתכנס לפעולה כאשר התהליך מפסיק את פעולתו. שגרות exitlist אלו יכולות לשחרר משאבים שיתכן ויידרשו על-ידי תהליכים אחרים. לדוגמה, תהליך יכול להחזיק במשאב מערכת כאשר הוא מופסק בצורה לא שגרתית. שיגרת exitlist של התהליך נכנסת לפעולה ויכולה לשחרר את משאב המערכת, כדי שתהליכים אחרים יוכלו להשתמש בו. אם לא כן, משאב זה לא יוכל להיות בשימוש. השימוש במנגנון זה גורם לכך, שהתהליך אינו מאלץ תהליכים אחרים לחכות לנצח למשאב כדי שיתפנה.

בנוסף לכך, שגרות exitlist משתמשות ב-DosExitList כדי להכניס לפעולה

שגרות exitlist אחרות השייכות לאותו תהליך. כל שיגרת exitlist חייבת לסיים את פעולתה מהר ככל האפשר, כדי שמערכת ההפעלה תוכל להשלים את הפעולות הדרושות להפסקת התהליך.

מישקי תכנות ל-THREADS

thread יכול ליצור ולהתחיל threads אחרים באותו תהליך. thread בתהליך יכול להשעות, או לחדש את פעולתו של thread מסוים בתוך אותו תהליך. thread יכול גם למנוע מ-threads אחרים מלהתבצע בזמן שחלק קריטי שלו מתבצע.

DosCreateThread ה-thread יכול ליצור ולהתחיל thread חדש השייך לאותו תהליך. ה-thread החדש מתבצע בצורה בלתי-מסונכרנת לביצוע של ה-thread הנוכחי. מספר הזיהוי (Thread ID) של ה-thread החדש מוחזר למבקש.

ה-thread המבקש מעביר מחוון באמצעות פקודת הסתעפות אל קבוצת פקודות שתקבל את השליטה ותתחיל את הביצוע של ה-thread החדש. המבקש מעביר גם מחוון למחסנית (stack) של ה-thread החדש. מערכת ההפעלה מכניסה את הערכים לאוגרי המעבד, יוצרת ומתחילה את ה-thread החדש באותה עדיפות ביצוע של התכנית המבקשת.

DosSuspendThread ה-thread המבקש משעה את פעולתו של thread אחר בתוך אותו תהליך על-ידי ציון מספר הזיהוי המסוים של אותו thread. ה-thread המושעה לא יבצע פקודות נוספות של היישום, אבל ייתכן והוא לא יפסיק את פעולתו מיד לאחר קבלת פקודת ההשעייה, אם הוא מתבצע באותו רגע ותופס משאבי מערכת.

DosResumeThread ה-thread המבקש מתחיל מחדש thread (השייך לאותו תהליך) שהושעה קודם לכן. הוא מציין את מספר הזיהוי המתאים של אותו thread.

DosEnterCritSec thread יכול למנוע ביצוע של threads אחרים הנמצאים בתהליך שלו. המערכת סופרת את מספר הפעמים שבוצעה קריאה לפונקציה זו (ערך בן 16 סיביות). דבר זה יכול להבטיח ש-thread אחר בתהליך לא יכנס למשאב שה-thread מבקש להשתמש בו בצורה סידרתית. קריאה זו אינה מונעת משיגרת הטיפול באותות של התהליך מלהשתלט על הביצוע כשדרוש.

DosExitCritSec הסרת ההשפעה של הפונקציה הקודמת DosEnterCritSec. אם בוצעו קריאות רבות ל-DosEnterCritSec, יש צורך לבצע את אותו מספר קריאות ל-DosExitCritSec, כדי לאפשר ל-threads האחרים בתהליך להתבצע.

מישקי תכנות לקביעת עדיפות ביצוע

עדיפות ביצוע של תהליך או thread מצביעה על חשיבותו, ומכאן - על הסיכוי שלו להתבצע. thread המפקח על-כך שערוץ תקשורת מסוים יפעל כשורה, חשוב הרבה יותר מאשר thread המעדכן את הזמן על המסך, כי למשתמש לא אכפת אם הזמן על המסך מתעדכן בדיוק מלא. עם זאת, למשתמש יפריע מאוד אם ערוץ התקשורת אינו פועל מכיון שה-thread האחראי עליו אינו יכול להגיב.

המישק לתכנות יישומים מאפשר ליישום לקבוע עדיפות ביצוע לגורמים אחרים במערכת. חשוב מאוד שיישום לא ינצל לרעה אפשרות זו וייקח לעצמו את עדיפות הביצוע הגבוהה ביותר במערכת. אם כל יישום יעשה זאת, המערכת לא תוכל לפעול כראוי.

אם למספר threads יש את אותה עדיפות ביצוע, OS/2 תקצה **פלח זמן** (timeslice) שווה לכל אחד מהם. זאת אומרת, שמערכת ההפעלה מאפשרת לכל thread להתבצע בפרק זמן מוגדר, בשיטה הנקראת **תזמון בשיטת הריצה המחזורית** (round robin scheduling). שיטה זו מאפשרת ל-threads רבים שיש להם אותה עדיפות ביצוע להתחלק במעבד בצורה הוגנת. למשתמש יש שליטה על פרק הזמן בעזרת פרמטר של ה-CONFIG.SYS הנקרא TIMESLICE.

אם ל-thread אין אפשרות להתבצע תוך פרק זמן מסוים, הוא מקבל "העלאה" בעדיפות הביצוע. למשתמש יש שליטה על פרק הזמן שעובר עד שמתבצעת ההעלאה בעדיפות, בעזרת פרמטר של ה-CONFIG.SYS הנקרא MAXWAIT.

העדיפות של ה-thread מורכבת מאחת משלוש רמות של עדיפות ביצוע ומתת-רמה הנלווית אליה. שלושת הרמות הן:

- קריטית מבחינת זמן ביצוע - Time Critical
- רגילה - Regular
- אדישה - Idle

לכל רמת עדיפות יש 32 תת-רמות (0-31). ככל שהמספר יותר גדול, כך עדיפות הביצוע גבוהה יותר. לרמה הקריטית מבחינת הזמן יש עדיפות ביצוע גבוהה יותר מאשר לרמה הרגילה, ולרמה הרגילה יש עדיפות על הרמה האדישה. OS/2 תבצע threads עם עדיפות ביצוע גבוהה לפני threads עם עדיפות ביצוע נמוכה.

כפי שהשם מרמז, הרמה הקריטית מכוונת ל-thread עם פעילות שצריכה להתבצע תוך פרק זמן קצר מאוד מהתרחשות של אירוע מסוים. אם הפעילות הקריטית מבחינת זמן אינה מתבצעת, היישום או סביבת העבודה יפסיקו לפעול. מערכת ההפעלה אינה משנה את תת-הרמה של ה-threads הנמצאים ברמה הקריטית מבחינת זמן הביצוע (תזמון קבוע). OS/2 מתזמנת את ה-threads הנמצאים ברמה הקריטית באותה תת-רמה בשיטת הריצה המחזורית.

הרמה האדישה משמשת ל-threads המתבצעים כאשר אין עבודה אחרת ברמה גבוהה יותר, והיא מאפשרת לנצל את כוח העיבוד במערכת בצורה אופטימלית. מערכת ההפעלה אינה משנה את תת-הרמה של ה-threads הנמצאים ברמה האדישה (תזמון קבוע). OS/2 מתזמנת את ה-threads הנמצאים ברמה האדישה באותה תת-רמה בשיטת הריצה המחזורית.

ברמה הרגילה רצים רוב ה-threads במערכת. מערכת ההפעלה משנה את תת-הרמה של ה-threads הנמצאים ברמה זו. השינוי בתת-הרמה תלוי בפעילות שה-thread מבצע וביתר הפעילויות המתבצעות במערכת. דוגמאות של פעילויות שיכולות להשפיע על תת-הרמה כוללות thread המבצע פעולות ק/פ ב-foreground, או thread שלא קיבל הזדמנות להתבצע.

DosSetPrty משנה את רמת עדיפות הביצוע ואת תת-הרמה הקשורה אליה עבור

היישויות הבאות:

- תהליך וכל ה-threads שלו. התהליך מזוהה על-ידי מספר זיהוי. ניתן לבחור גם בתהליך הנוכחי.
- thread מסוים בתוך התהליך הנוכחי, או ה-thread הנוכחי.
- תהליך (וכל ה-threads שלו) וכל הצאצאים שלו (וכל ה-threads שלהם). התהליך הנבחר חייב להיות התהליך הנוכחי, או צאצא של התהליך הנוכחי.

המשתמש יכול לבחור בין שינוי של רמת העדיפות לבין שינוי של תת-הרמה, או בשינוי של שתיהן. השינוי המבוקש בתת-הרמה צריך להיות יחסי לערך הנוכחי של תת-הרמה. לדוגמה, המבקש רוצה להעלות את תת-הרמה הנוכחית ב-5 דרגות. אם הוא משנה את רמת העדיפות, עליו לציין רמת עדיפות מסוימת. לדוגמה, המבקש רוצה לשנות את רמת העדיפות הנוכחית לרמה הקריטית מבחינת הזמן. כאשר נוצר thread, הוא יורש עדיפות ביצוע הזהה לזו של אביו.

DosGetPrty מאפשר ל-thread לקבל את רמת העדיפות ואת תת-הרמה של ה-thread שהתחיל אותו, או של thread מסוים (על-פי מספר הזיהוי) בתהליך.

תקשורת בין תהליכים

כאשר יישום מבוסס על תהליך אחד ויחידות ביצוע רבות, קל מאוד לראות איך ה-threads השונים מתקשרים בינם לבין עצמם. כולם מתחלקים באותם משאבים (כמו זיכרון, למשל). במצב כזה חשוב להבטיח ש-threads שונים לא יפריעו זה לזה.

אנו יודעים כבר איך מישק התכנות מאפשר ל-threads לשלוט על הביצוע של threads אחרים באותו תהליך. הם יכולים להשעות אותם, להתחיל אותם מחדש ולהפסיק את הביצוע של שאר ה-threads.

כאשר יישום מורכב מתהליכים רבים, או כאשר תהליך מיישום אחד צריך להתקשר עם תהליך מיישום אחר, צריך להתחשב בכך שהתהליכים אינם מחלקים ביניהם את משאבי המערכת. מערכת ההפעלה חייבת לספק פונקציות מיוחדות המאפשרות לתהליכים להעביר, או להתחלק בנתונים זה עם זה. הפונקציות הן:

- זיכרון משותף (shared memory).
- צינורות (pipes).
- תורים (queues).

למדנו כיצד תהליך יוצר סגמנט שניתן להתחלק בו עם תהליך אחר. צינורות ותורים הם שני מנגנונים נוספים להעברת נתונים או להתחלק בהם.

בנוסף לכך, תהליכים יכולים לשלוט בגישה או לפנות לסירוגין למשאבים משותפים, או לפונקציות. הם גם יכולים להעביר את עובדת התרחשותם של אירועים מסוימים זה לזה. הדבר נעשה בעזרת המנגנונים הבאים לתקשורת בין

תהליכים פנימיים:

- אותות (signals).
- אתתים (semaphores).

אות מבוצע על-ידי אחת היישויות במערכת (תהליך או thread). האתת הינו מקום בזיכרון, אשר נקבע על-ידי אחת היישויות האלו לצורך העברת הודעות על אירועים המתבצעים במערכת.

צינורות

שני תהליכים ויותר מאותה משפחה יכולים להתקשר באמצעות צינור. תהליכים מאותה משפחה הם כל התהליכים שירשו מאפיינים מאותו "מייסד" של שושלת התהליכים. לדוגמה, שני תהליכים שהם בנים של אותו תהליך אב, שייכים לאותה משפחה. אם תהליך יוצר צינור, כל הבנים שהאב יצר יכולים לרשת את היכולת לכתוב לצינור, או לקרוא ממנו.

תקשורת באמצעות צינור דומה לכתיבה לקובץ וקריאה ממנו, וההבדל היחיד הוא בכך שצינור נמצא בזיכרון ולא על הדיסק. כאשר הצינור נוצר, היוצר קובע את גודלו שיכול להגיע עד 64KB.

התהליך קורא נתונים מהצינור לפי הסדר של "ראשון נכנס ראשון יוצא" (FIFO). אין דרך לשנות את סדר הנתונים בצינור ויותר מכך, ברגע שתהליך קרא נתונים מהצינור, הוא אינו יכול לקרוא אותם שוב. מערכת ההפעלה עורכת מעקב על נתונים ועל שטח פנוי בצינור. אם thread כותב לצינור והצינור מלא, מערכת ההפעלה מאלצת אותו לחכות עד אשר יווצר מקום פנוי בצינור עבור הנתונים שלו.

הנתונים שהתהליכים מעבירים ביניהם מועתקים למעשה פעמיים. בפעם הראשונה בין הכותב לצינור ובפעם השניה - בין הצינור לקורא. שיטה זו אינה יעילה אם מועברים נתונים רבים בין תהליכים.

DosMakePipe תהליך יוצר את הצינור וקובע את גודלו. ניתן לקרוא מהצינור ולכתוב אליו בעזרת שגרות ק/פ סטנדרטיות, DosRead ו-DosWrite. שני מקשרים מוחזרים למבקש: אחד ל-DosRead ואחד ל-DosWrite.

לבנים של תהליך האב יש זכות לכתוב לצינור ולקרוא ממנו, אם הם ירשו את הזכות הזאת והם משתמשים במקשרים הנכונים. הגישה למקשרים מופסקת בעזרת שיגרת הק/פ לקבצים, DosClose. הצינור מתבטל כאשר כל התהליכים המשתמשים בצינור מוציאים DosClose.

תורים

תור הוא מנגנון נוסף שהתהליכים יכולים להשתמש בו כדי להעביר נתונים או להתחלק בהם. שיטת התור טובה וגמישה יותר מאשר שיטת הצינור.

כאשר שני תהליכים מתקשרים באמצעות תור, המידע אינו עובר בין שני התהליכים. במקום המידע מועבר **האיבר של התור** (queue element) הכולל מחוון לנתונים וערך המציין את אורכם. השימוש בתור יעיל יותר מאשר

השימוש בצינור, כי חוסכים את העתקת הנתונים. במקום העתקה, מועבר המידע הדרוש כדי למצוא את הנתונים המנוהלים באמצעות מנגנון התורים.

לכל התהליכים המשתמשים בתור חייבת להיות אפשרות מיעון לסגמנטים המכילים נתונים. כאשר תהליך קורא את האיבר של התור ומנסה להכנס לסגמנט המכיל את הנתונים, הוא לא יוכל לעשות זאת כל עוד לא קיבל רשות ואפשרות להכנס לאותו סגמנט. זאת אומרת, שיש צורך להשתמש בתור בשילוב עם זיכרון משותף.

האיברים של התור יכולים להקרא מהתור ולהמחק לאחר מכן. התהליך יכול גם לקרוא את האיברים של התור לפי סדר "ראשון נכנס ראשון יוצא", לפי סדר של "אחרון נכנס ראשון יוצא" (LIFO), או לפי סדר עדיפות כלשהו. אם נקבע סדר עדיפות לאיברים, הכותב לתור מפרט את העדיפות היחסית של האיבר (0-15). מסיבה זו ברור שתורים גמישים בהרבה מאשר צינורות, שבהם מנגנון הקריאה ההרסני (כי אינו ניתן לקריאה חוזרת) לפי FIFO הינו הדרך היחידה לקרוא מתוך הצינור.

התהליך היוצר (הבעלים) את התור הינו היחיד שיכול לקרוא ממנו את האיברים. גם כל ה-threads שלו יכולים לקרוא מהתור. תהליכים אחרים יכולים לקבל הרשאה לכתוב לתוך התור, וגם כל ה-threads בתוך התהליכים הכותבים יכולים לעשות זאת. לסיכום, לתור יש קורא אחד וכותבים רבים.

התהליך שקורא מהתור מבצע את הפעולות הבאות:

- יוצר את התור - DosCreateQueue.
- קורא איבר מהתור - DosReadQueue.
- מציץ אל התור - DosPeekQueue.
- סוגר את התור - DosCloseQueue.

התהליך שכותב לתור מבצע את הפעולות הבאות:

- פותח את התור - DosOpenQueue.
- כותב איבר לתור - DosWriteQueue.
- סוגר את התור - DosCloseQueue.

DosCreateQueue יוצר את התור. המבקש (התהליך שהינו הבעלים של התור) נותן לתור שם, הנראה בדיוק כמו שם-קובץ. השם חייב להתחיל ב-\QUEUES\. כאשר תהליך אחר רוצה לפתוח את הקובץ לקריאה, הוא משתמש בשם זה.

התהליך המבקש בוחר בסדר קריאת האיברים מתוך התור. האפשרויות הן FIFO, LIFO, או לפי סדר עדיפות.

התהליך המבקש מקבל מקשר (handle) שישמש מישקי תכנות אחרים הקשורים בתור. התהליך המבקש הינו היחיד שיכול לקרוא את האיברים מתוך התור.

DosOpenQueue פותח את התור לכתיבה. חייבים ליצור תור לפני שפותחים אותו לכתיבה. ברגע שנוצר תור הוא מקבל שם, אשר משמש תהליכים אחרים הרוצים לפתוח אותו. השם צריך להיות דומה לשם קובץ, וצריך להתחיל ב-\QUEUES\.

התהליך המשתמש במישק זה מקבל את מספר הזיהוי של הבעלים של התור ואת המקשר המאפשר כתיבה לתור. לאחר שתהליך פתח את התור, הוא רשאי לכתוב לתוכו.

DosWriteQueue מוסיף איבר לתור המזוהה על-ידי מקשר. איבר התור מכיל את הפרטים הבאים:

- הכתובת של הנתונים.
- האורך של הנתונים.
- מלה נוספת המשמשת לתיאור המידע שהאיבר מצביע עליו. המשמעות של המלה חייבת להקבע מראש על-ידי התהליך הקורא והתהליכים הכותבים באותו תור.

המשתמש במישק זה מפרט את העדיפות של האיבר החדש. איברים עם אותה עדיפות מועברים לפי סדר FIFO.

DosReadQueue מאפשר לכל thread של התהליך שהינו הבעלים של התור לקרוא ממנו איברים. מקשר התור (handle) מזהה את התור שיש צורך לקרוא ממנו. האיבר מוצא מהתור לאחר שהוא נקרא.

ה-thread המשתמש במישק זה מקבל את הפרטים הבאים:

- האיבר (גודל, כתובת הנתונים ומלת המידע הנוספת).
- מספר הזיהוי של התהליך שכתב את האיבר לתור.
- העדיפות של האיבר.

הקורא של התור יכול לעקוף את סדר הקריאה הרגיל של האיברים על-ידי שימוש ב-DosPeekQueue. האיבר הקודם שנבחן בעזרת DosPeekQueue יכול עדיין להקרא בעזרת DosReadQueue.

ה-thread המשתמש במישק זה יכול לעשות את אחד מהדברים הבאים:

- להמתין עד אשר יהיה לפחות איבר אחד בתור.
- לא להמתין, אפילו אם אין איבר אחד בתור.

אם המשתמש במישק זה בוחר באפשרות ההמתנה, הוא חייב לציין את המקשר לאתת. אם threads רבים של בעל התור קוראים מהתור, עליהם להשתמש באותו מקשר לאתת. אנו נדון באתתים מאוחר יותר בפרק זה. האתת מאפשר לתהליך לדעת מתי נכתבים נתונים לתוך התור ולפעול בהתאם.

DosPeekQueue מבצע את אותה פעולה כמו DosReadQueue פרט לדבר אחד. התהליך המציץ (שחייב להיות בעל התור) משתמש במישק זה כדי להציץ באיבר מבלי להוציא אותו מהתור. בכך הוא מאפשר לבעל התור לסרוק את האיברים עד שהוא מוצא איבר מסוים. לאחר שמצא, הוא משתמש ב-DosReadQueue כדי להוציא את האיבר המסוים הזה מהתור.

DosQueryQueue מאפשר לתהליך הקורא ולתהליכים הכותבים לדעת את מספר האיברים הנמצאים כרגע בתור. המקשר לתור מזהה את התור.

DosPurgeQueue בעל התור מוחק את את כל האיברים הנמצאים בתור ומותיר אותו ריק.

DosCloseQueue מבטל את הרשות של תהליך להכנס לתור. אם בעל התור סוגר את התור, אז התור נמחק. בנקודה זו, כל אחד מהכותבים לתור מקבל קוד המודיע שהופסקה פעולתו.

מנגנון התורים הינו בעל עוצמה רבה ומאפשר לתהליכים, אשר יש להם זיכרון משותף, להעביר נתונים בינם לבין עצמם. אפשר להעביר ביעילות כמויות גדולות של נתונים כי אין צורך להעתיק אותם; במקום העתקה מועבר איבר תור מתהליך אחד לחברו. הסדר שבו תהליך מעביר את האיברים הינו גמיש ביותר. התהליך הקורא יכול לקרוא (להציץ) איברים מבלי להוציאם מהתור.

אותות

ראינו כיצד תהליכים יכולים להתקשר בינם לבין עצמם על-ידי משלוח וקבלה של נתונים. בנוסף לכך דרוש מנגנון שיאפשר לתהליך להודיע לתהליך אחר לעשות משהו **כעת**, בדומה לדרך הפעולה של פסקים בתוכנה. פסק בתוכנה מכניס לפעולה מיידית שיגרת טיפול מתאימה.

תהליך מייעד שיגרת טיפול לאירוע הנקרא **אות** (signal). לאחר שבוצעה הקריאה לשיגרת הטיפול היא מבוצעת על-ידי ה-thread הפותח של התהליך, אשר נוצר כחלק מהתהליך בעזרת DosExecPgm. התהליך יכול להשתמש באותות לדברים אחרים מאשר קבלה של אירועים יוצאי דופן. אם הוא רוצה לעשות כן, הוא אינו יכול להשתמש ב-thread הפותח לביצוע עבודה רגילה. אפשר ליצור threads נוספים בעזרת DosCreateThread כדי לבצע את העבודה הרגילה. מערכת ההפעלה או התהליך יכולים לשלוח אות לתהליך מסוים. כאשר השליטה עוברת לשיגרת הטיפול באות, מועברים אליה הכתובת וסוג האירוע שהתרחש. השולח של אות-הסימון (flag) יכול גם להעביר מידע לשיגרת הטיפול באות-הסימון המסוים.

התהליך יכול לייעד שיגרת טיפול באותות לסוגי האירועים הבאים:

- מקשים Control+Break לחוצים (SIGBREAK).
- מקשים Control+C לחוצים (SIGINTR).
- הפסקת התכנית (SIGTERM).
- אות-סימון A (SIGPFA).
- אות-סימון B (SIGPFB).
- אות-סימון C (SIGPFC).

אם התהליך אינו יוצר שגרות לטיפול באותות SIGBREAK, SIGTERM, או SIGINTR, מערכת ההפעלה תפסיק אותו אם אחד מהאותות האלה יישלח אליו. תהליך יכול ליצור שגרות טיפול באותות אלה כדי שהם לא יפסיקו אותו.

תהליך יכול לשלוח לתהליך אחר כל אחד מאותות הסימון בעזרת המישק DosFlagProcess. תהליך יכול להשתמש במישק DosSendSignal כדי להעביר לתהליך אחר אות דימוי של Control+Break, או Control+C.

פונקציות הטיפול באותות הן מנגנונים בעלי עוצמה רבה. תהליכים יכולים להשתמש בהן כדי לבצע פקודה מיוחדת כאשר אחד מהאירועים הנ"ל מתרחש. התהליך יכול לשמור במצב אדיש (רדום) את ה-thread ההתחלי שלו, ואז הוא יוכל לטפל ללא קושי בכל אחד מהאותות שהוא ירצה לקלוט כדי לבצע את עבודתו.

הפונקציה DosKillProcess משמשת למשלוח אות SIGTERM לתהליך אחר.

DosSetSigHandler מאפשר לתהליך לפרט את הפעולה שתתבצע לכשיתקבל אות

מסוים. התהליך יכול לבקש שאחת מבין הפעולות הבאות תתבצע:

- ברירת המחדל של המערכת תתבצע כשיתקבל אות.
- יש להתעלם מהאות.
- שיגרת הטיפול באותות שהוגדרה על-ידי התהליך תפעל כאשר יתקבל אות.
- אם תהליך אחר מנסה לשלוח אות לתהליך זה, הוא יקבל הודעת שגיאה.
- שיגרת הטיפול באות מודיעה למערכת ההפעלה שהיא סיימה לעבד את האות והיא מוכנה לעבד אות נוסף.

כאשר תהליך מתקין שיגרת טיפול לאותות, מועברים אליו הכתובת של שיגרת הטיפול הקודמת והפעולה שהיא עמדה לנקוט בעת קבלת האירוע. אם התהליך אינו רוצה ששיגרת הטיפול הנוכחית תהיה פעילה, הוא יכול לשחזר את שיגרת הטיפול הקודמת בעזרת מידע זה.

DosHoldSignal מאפשר לתהליך להשעות את העיבוד של האותות לפרק זמן קצר ביותר. לדוגמה, ספריית השגרות של הקישור הדינמי יכולה להקצות משאב לתהליך, אך היא אינה יכולה לטפל בהפסקת פעולתו של התהליך לפרק זמן קצר. פונקציה זו מאפשרת לתהליך להפסיק ולחדש את הטיפול באותות. פעולות אלו יכולות להיות מקוננות (nested).

המערכת תזכור שהאות הופק ותעביר אותו לתהליך מיד לאחר שהוא יחדש את עיבוד האותות. פרק הזמן שבו לא יבוצע עיבוד האותות צריך להיות קצר ככל האפשר, כשם שפרק הזמן שבו לא יבוצע העיבוד של פסקי החומרה צריך להיות קצר ככל האפשר.

DosFlagProcess תהליך יכול לשלוח אות-סימון (A, B או C) לתהליך אחר, או לתהליך אחר ולכל הצאצאים שלו. זאת אומרת ששיגרה אחת (או יותר) לטיפול באותות תכנס לפעולה. המשתמש יכול לציין את המשתנה שתקבל שיגרת הטיפול באותות של תהליך המטרה. אות לא יטופל אם תהליך המטרה לא התקין שיגרה לטיפול באותו סוג אות. תהליך המטרה יכול גם לקבוע שתתקבל הודעת שגיאה כאשר מנסים לשלוח את סוג האות המסוים באמצעות DosSetSigHandler.

המשתמש במישק זה מעביר למערכת את מספר הזיהוי של התהליך הרצוי ואת סוג האות.

DosSendSignal התהליך שולח Control-C או Control-Break לקובץ האחרון שהופעל בעץ של אותו תהליך שזוהה בעזרת מספר הזיהוי המתאים. המערכת מחפשת שיגרת טיפול באותות על-ידי סריקה של תהליכי האב, עד אשר היא מוצאת את השיגרה המתאימה.

אתתים

אתתים (semaphores) הינם מנגנון התקשרות חשוב ביותר. הם מאפשרים ל-threads להשתמש במשאב או פונקציה בצורה סידרתית. אתתים מאפשרים ל-thread לשלוח אות ל-thread אחר כדי להודיע לו על התרחשות של אירוע. threads של אותו תהליך או threads של תהליכים שונים יכולים להשתמש באתתים. כלומר, אתתים יכולים לשמש לתקשורת בתוך התהליך, או לתקשורת בין תהליכים שונים.

יש שני סוגים של אתתים:
- אתתי מערכת (System semaphores).
- אתתי RAM (RAM semaphores).

אתת מערכת חייב להיווצר בעזרת DosCreateSem. התהליך היוצר נותן לאתת שם, ובכך הוא מאפשר לתהליכים אחרים להכנס אליו. כאשר תהליך יוצר או פותח אתת מערכת, הוא מקבל אליו מקשר של אתת מערכת. התהליכים משתמשים במישקי תכנות אחרים לאתתים הנמצאים במערכת ההפעלה.

אתתי מערכת מעמיסים את מערכת ההפעלה יותר מאשר אתתי RAM. מערכת ההפעלה מספקת מנגנון פתיחה (DosOpenSem), אשר מאפשר לתהליכים אחרים להכנס לאתת המערכת מבלי שיהיה להם זיכרון משותף. במקום זה, התהליכים השונים צריכים להשתמש בשם של האתת. מערכת ההפעלה מנהלת גם את הפסקת הפעולה של תהליך שבבעלותו נמצא אתת המערכת. תהליכים אחרים אינם צריכים להמתין עד אשר התהליך שפעולתו הופסקה ישחרר את אתת המערכת, מכיון שמערכת ההפעלה מפקחת על האתת.

אתת ה-RAM הינו למעשה מלה נרדפת לזיכרון המערכת, והמקשר אליו הינו הכתובת של המקום בזיכרון המכיל את האתת. אין צורך ליצור או לפתוח אתת RAM, אבל יש לדאוג לכך שהוא יהיה ידוע. לכל התהליכים המשתמשים באתת צריכה להיות אפשרות גישה לסגמנט הזיכרון שבו הוא נמצא.

אתתי ה-RAM דורשים הרבה פחות פיקוח של המערכת מאשר אתתי המערכת. השימוש הטוב ביותר בהם יהיה בין threads שונים של אותו תהליך. המערכת אינה נוקטת בשום פעולה כדי לנהל את אתתי ה-RAM. לדוגמה, אם הופסקה פעולת תהליך שהינו הבעלים של אתת RAM מבלי שהוא שחרר את האתת, מערכת ההפעלה לא תודיע לתהליך אחר שהבעלים של אותו אתת הופסק, ועל-כן לא יוכל לשחרר אותו לעולם.

מישקים לאתתי מערכת

תהליכים יכולים ליצור אתתי מערכת, או לקבל אישור כניסה לאתת מערכת שכבר קיים.

DosCreateSem מאפשר לתהליך ליצור את אתת המערכת ולתת לו שם. השם הוא שם קובץ הנרשם לפי הכללים ומתחיל ב-\SEM\. התהליך מקבל מקשר לאתת שאיתו הוא יכול להשתמש עם מישקים אחרים לאתתים.

התהליך היוצר את אתת המערכת קובע אם הוא יהיה **בלעדי** (exclusive), או לא. השימוש הטוב ביותר באתתי מערכת בלעדיים הוא כאשר יש צורך להשתמש בפונקציות שנדרשת בהן בעלות על האתת. **thread** משיג את הבעלות על אתת המערכת על-ידי כך שהוא משתמש במישק DosSemRequest. אם אתת המערכת נוצר עם מאפיין של בלעדיות, רק thread אחד יכול להיות הבעלים שלו, ורק הוא יוכל לשנות את המצב של אותו אתת. הדבר מונע מ-threads אחרים לשחרר את האתת מידי הבעלים שלו. רק ה-thread שהוא הבעלים של אתת מערכת בלעדי יכול לשחרר אותו.

אם אתת המערכת משמש לאיתות, יש צורך ליצור אותו כלא-בלעדי, ובכך לאפשר לכל thread לשנות את המצב שלו. ברוב המצבים שבהם יש צורך לשלוח אות משתמשים באפשרות זו.

DosOpenSem מאפשר לתהליך להכנס לאתת מערכת שנוצר על-ידי תהליך אחר. לשם כך עליו לקבל את השם שניתן לאתת בזמן יצירתו באמצעות המקשר לאתת. התהליך שיצר את אתת המערכת אינו צריך להשתמש בפונקציה זו.

כאשר תהליך יוצר תהליכים חדשים (DosExecPgm), הם יורשים את המקשרים המאפשרים את פתיחת האתתים. תהליכי הבנים לא יהיו הבעלים של האתתים, גם אם תהליך האב היה הבעלים של אתת כזה בזמן שהוא יצר אותם.

DosCloseSem מיועד לתהליך שאינו רוצה להשתמש עוד באתת המערכת. התהליך מעביר למישק התכנות את המקשר של אתת המערכת שאותו הוא רוצה לסגור. אתת המערכת נמחק מהזיכרון כאשר כל התהליכים השתמשים באותו אתת מערכת מוציאים הוראה כזאת.

אם תהליך מופסק ואינו סוגר אתת מערכת, המערכת עושה זאת באופן אוטומטי. אם מופסק תהליך בעודו משתמש באתת המערכת, המערכת מודיעה זאת לכל ה-threads של התהליכים האחרים הממתינים לאתת.

מישקים להשגת בעלות על אתת

מישקים אלה מתאימים הן עבור אתת מערכת והן עבור אתתים של ה-RAM. הפונקציות להשגת הבעלות על אתתים משמשות בעיקר אתתי מערכת בלעדיים. מרגע ש-thread השיג בעלות על אתת מערכת בלעדי, הוא היחיד שיכול לשנות את מצבו.

DosSemRequest מאפשר ל-thread לטעון לבעלות על אתת המזוהה על-ידי המקשר לאתת. אם האתת אינו בבעלות של אף אחד, ה-thread המשתמש במישק זה יהיה מרגע זה הבעלים של האתת. אם האתת נמצא כבר בבעלות של אחר, ה-thread הנוכחי יכול לבחור באחת מדרכי הפעולה הבאות:

- להמשיך להתבצע ללא האתת.
- לחכות עד אשר הוא יוכל להשיג את הבעלות על האתת.
- להמתין פרק זמן מוגדר מראש, ואם עד אז האתת לא יהיה פנוי - לוותר עליו.

אם thread מחכה עד אשר ישיג את הבעלות על האתת, מערכת ההפעלה תבצע בדיקת-מצב של האתת כאשר היא תנסה לבצע את ה-thread. אם האתת לא נמצא במצב המתאים, מערכת ההפעלה לא תתחיל את הביצוע. המערכת אינה בוחנת שוב אם האתת שוחרר, ועל-כן ה-thread לא יבוצע גם אם האתת שוחרר לאחר שהתקבלה תשובה שלילית בעת הבדיקה. לכן, הדבר מאפשר ל-thread אחר שמתבצע באותו זמן להשתלט על האתת. האתת צריך להיות פנוי בזמן שהמערכת מנסה לבצע (dispatch) את ה-thread הממתין.

אם האתת הוא אתת מערכת ונוצר עם מאפיין של בלעדיות, הבעלים יכול לקרוא ל-DosSemRequest, בצורה רקורסיבית. המערכת עורכת מעקב אחר מספר הפעמים שה-thread ביקש את האתת. כאשר האתת נמצא בבעלות של מישהו, threads אחרים אינם יכולים לבצע פונקציות שיכולים לשנות את מצבו (כמו DosSemClear, DosSemSet וכו').

מישק זה משמש עבור אתתי RAM ואתתי מערכת לא בלעדיים, למטרות של שיגור אותות.

DosSemClear בעזרת מישק זה משחרר ה-thread את האתת. thread משיג את הבעלות על האתת על-ידי קריאה ל-DosSemRequest. אם אתת המערכת הוא בלעדי, ו-DosSemRequest בוצע מספר פעמים, יש לבצע את DosSemClear אותו מספר פעמים כדי שהאתת יהפוך לחופשי. אם אתת המערכת הוא לא-בלעדי, כל thread יכול להשתמש במישק זה כדי לשחרר את האתת.

אפשר להשתמש במישק זה יחד עם המישקים DosSemWait, DosSemSetWait ו-DosMuxSemWait כדי לנקות את האתת למטרות של איתות.

מישקים להעברת אותות באמצעות אתתים

מישקים מתאימים הן לאתתי מערכת והן לאתתי RAM. אתתי מערכת בלעדיים אינם משמשים בד"כ להעברת אותות.

אפשר להשתמש במנגנון של DosSemRequest ו-DosSemClear להעברת אותות באמצעות אתתי מערכת לא-בלעדיים ואתתי ה-RAM. במקרה כזה ה-thread ימתין כדי שהקריאה ל-DosSemRequest תסתיים. הוא יחכה לאירוע שינקה את האתת ורק אז הוא יוכל להמשיך להתבצע. אין זה משנה מי הבעלים של האתת. כל thread יכול לשלוח אות ל-thread הממתין על-ידי שימוש ב-DosSemClear.

מישקי התכנות המתוארים מטה משמשים כדי לקבוע את השימוש באתתים ולהמתין להם שישתחררו. DosSemClear משמש לשחרור אתתים.

DosSemSet קובע שהאתת ישמש להעברת אותות (signalling). פעולת מישק זה מנוגדת ל-DosSemClear, המשחרר את האתת מפעילות של העברת אותות.

DosSemWait ה-thread המשתמש במישק זה ממתין עד אשר האתת יהיה חופשי, או עד אשר חולף פרק זמן מסוים. אם האתת כבר חופשי, אין צורך להמתין לו. המשתמש במישק זה יכול לקבל את השליטה מיד, אם האתת נמצא בבעלות של מישהו, או להמתין עד אשר האתת ישוחרר. מישק זה אינו מאפשר לקבל בעלות על האתת.

בדומה ל-DosSemRequest, מערכת ההפעלה בודקת את מצב האתת כדי לראות אם הוא פנוי. המצב של האתת חשוב רק כאשר מערכת ההפעלה מנסה לחדש את הביצוע של ה-thread.

DosSemSetWait מבצע פעולה דומה לזו של DosSemWait, פרט לכך שניתן לקבוע את הבעלות בעזרת מישק זה. ה-thread המשתמש במישק זה חייב להמתין שהאתת יתפנה, אלא אם הוא מבקש לקבל את השליטה מיד, בלי קשר למצב האתת.

בדומה ל-DosSemWait, מערכת ההפעלה בודקת את מצב האתת כדי לראות אם הוא פנוי. המצב של האתת חשוב רק בנקודה שבה מערכת ההפעלה מנסה לחדש את הביצוע של ה-thread.

DosMuxSemWait ה-thread המשתמש במישק זה ממתין עד אשר יתפנה אחד מתוך קבוצה של אתתים. המשתמש במישק זה בוחר פרק זמן מסוים, כך שגם אם אף לא אחד מהאתתים יתפנה, המערכת עדיין תוכל לבצע את ה-thread. המשתמש יכול לבחור לקבל את השליטה מיד גם אם אף לא אתת אחד התפנה, או שהוא יכול לקבל את השליטה רק כאשר אחד מהאתתים מתפנה.

ישנו הבדל משמעותי בין פונקציה זו לפונקציות הקודמות. המשתמש במישק זה יכול לפרט רשימה של אתתים שצריכה להיות בפיקוח. אם אחד מהם מתפנה, אפשר לבצע את ה-thread. בנוסף לכך, הבדיקה של כל אתת נעשית בצורה של "הפעלת ספר". זאת אומרת, שאם האתת התפנה והשתלטו עליו שוב (לפני שמערכת ההפעלה ניסתה לבצע את ה-thread שקרא ל-DosMuxSemWait), עדיין ניתן לבצע את ה-thread הממתין.

השירותים של השעון הפנימי

מספר של מישקי תכנות ב-OS/2 נותנים שירותים הקשורים לפעולות הבאות:

- קבלה וקביעה של זמן ותאריך.
- המתנה לפרק זמן מוגדר.
- התחלה והפסקה במרווחי זמן ("שעוני עצר").

רוב המישקים מספקים קלט ופלט ביחידות של אלפיות השניה. חשוב לציין שהדיוק של המערכת לא מגיע לכך. מערכת ההפעלה משתמשת בשעון פנימי שפועל בתדירות של 32Hz. זאת אומרת, שמערכת ההפעלה עוקבת אחר הזמן בערך כל 1/32 של השניה. פרק זמן זה נקרא **תקתוק** (tick) השעון. הדיוק של מרווחי זמן יכול להגיע עד ±2 תקתוקים. זאת אומרת שכרגע דיוק הזמן לצורך ביצוע הפונקציות שנדונו הוא בערך 50 אלפיות השניה. אפילו אם ה-thread מוכן לביצוע בדיוק של 50 אלפיות השניה, אין זה מחייב את מערכת ההפעלה לבצע אותו מיידית.

השעון הפנימי מונה במדויק שעות, דקות ושניות, אך הוא אינו מונה במדויק אלפיות השניה בפרק זמן ארוך. תכניות העורכות מעקב אחר פרקי זמן ארוכים צריכות להתחשב בעובדה זו.

DosGetDateTime ה-thread המשתמש במישק זה מקבל את הפרטים הבאים:

- זמן בשעות, דקות, שניות, ומאיות השניה.
- תאריך על פי חודש, יום ושנה.
- איזור זמן ביחס לזמן הכללי (Universal Time).
- היום בשבוע.

DosSetDateTime מאפשר ל-thread המשתמש בפונקציה זו לקבוע את פרטי המידע המתקבלים על-ידי DosGetDateTime. אין אפשרות לקבוע את היום בשבוע.

DosSleep מאפשר ל-thread להשעות את הביצוע של עצמו לפרק זמן מוגדר מראש. פרק הזמן נמדד באלפיות השניה והוא מעוגל לתקתוק השעון הבא.

המשתמש בפונקציה זו חייב לקחת בחשבון שהדיוק של הפונקציה הוא בטווח של 2 תקתוקים. שיקולי תזמון לפי עדיפות יכולים למנוע מ-thread להתבצע למרות שתקופת ההשעיה עברה.
תכניות אינן צריכות להשתמש בפונקציה זו למטרות של מעקב אחר זמן.

DosTimerAsync הוא המקביל האסינכרוני של DosSleep. ה-thread המשתמש בפונקציה זו מעביר מקשר לאתת המערכת וממשיך להתבצע. כאשר פרק הזמן

המוגדר עובר, המערכת מנקה את האתת. על-ידי קביעת האתת לפני השימוש בפונקציה זו, התהליך יכול לגלות שפרק הזמן עבר על-ידי כך שהוא מקבל הודעה שהאתת אופס. המקשר לשעון העצר מוחזר למשתמש במישק זה, והוא משמש לעצור את שעון העצר לפני שהוא סיים את פעולתו.

DosTimerStart מקביל ל-DosTimerAsync, פרט לכך שבמקום לבקש הודעה על פרק זמן יחיד שעובר, הוא מבקש הודעה בכל פרק זמן מסוים.

לאחר שהמערכת מאפסת את האתת, היא מתחילה למדוד בצורה אוטומטית פרק זמן נוסף. אם התהליך רוצה לבצע פעולה כלשהי לאחר שיחלוף פרק זמן מסוים, הוא חייב לקבוע שוב את הבעלות על האתת. אם התהליך לא יקבע שוב את הבעלות לפני שפרק הזמן הזה חלף, הוא יחמיץ מרווח זמן אחד לפחות.

DosTimerStop מיועד לעצור את שעון העצר שהותחל על-ידי DosTimerAsync, או על-ידי DosTimerStart. המקשר לשעון, החוזר עם כל אחת מהפונקציות הנ"ל, משמש לעצירת השעון לפני שחלף פרק הזמן שהוגדר מראש. לאחר עצירת השעון יעברו האתתים ששימשו את הפונקציות האלו למצב לא מוגדר.

סגמנט מידע מקומי וגלובלי

מערכת ההפעלה מחזיקה שני סגמנטים של מידע (infoseg): **סגמנט מידע מקומי וסגמנט מידע גלובלי**. הסגמנט הגלובלי מכיל מידע אשר ישים לכל המערכת, והסגמנט המקומי מחזיק מידע עבור כל תהליך במערכת. יישומים אשר צריכים להשתמש במידע זה צריכים להיות ערים לכך, שסגמנט המידע יכול להתעדכן בשעה שהם קוראים אותו.

תהליך קורא את סגמנט המידע הגלובלי, או את סגמנט המידע המקומי שלו, על-ידי שימוש במישק התכנות **DosGetInfoSeg**.

סגמנט המידע המקומי מכיל את המידע הבא:

- מספר הזיהוי של התהליך הנוכחי.
- מספר הזיהוי של תהליך האב של התהליך הנוכחי.
- רמת עדיפות הביצוע ותת-הרמה של ה-thread הנוכחי.
- מספר הזיהוי של ה-thread הנוכחי.
- מספר הזיהוי של ה-session שהתהליך הנוכחי הוא חלק ממנו.

סגמנט המידע הגלובלי מכיל את המידע הבא:

- זמן ותאריך.
- מספר הגירסה הנוכחית של המערכת.
- פרמטרים שהמערכת משתמשת בהם לתזמון הביצוע של ה-threads.
- מספר הבלוק שהמערכת הותחלה ממנו.
- מספר הזיהוי של התהליך הנמצא ב-foreground.
- האם קיימת במערכת סביבת ביצוע של DOS.

פרק 4

עקרונות תכנות

פרק זה מיועד לקורא הרוצה לרדת לפרטים הטכניים הקשורים בניצול הפונקציות של OS/2. בפרק זה נראה מספר תכניות פשוטות המדגימות את השימוש במישק לתכנות יישומים (API). נתחיל בסקירה של הכלים הדרושים כדי לבנות תכנית לסביבת ה-OS/2 ולאחר מכן נציג מספר דוגמאות תכנות לשימוש ב-API.

לרשות המתכנת צריכים להיות תחנת עבודה עם OS/2, **ערכת כלים** (OS/2 Toolkit), מהדר ו/או אסמבלר. בפרק 1 ניתן למצוא את רשימת השפות הנתמכות על-ידי יבמ ב-OS/2. המהדרים, האסמבלרים, וכלים אחרים המסופקים עם שפות אלו הם **יישומי משפחה** (family applications) הניתנים לביצוע ב-OS/2 או ב-DOS 3.3. מוצרים אלה יכולים להפיק פקודה שתתקבל על-ידי כל אחת ממערכות ההפעלה.

הדוגמאות שנשתמש בהן בספר זה כתובות בשפת C. בחרנו ב-C מכיון שהיא הופכת לשפה מקובלת, ומכיון שסוגי הנתונים שלה, מבני הבקרה והספריות הסטנדרטיות לזמן ריצה איפשרו לנו להעביר בצורה היעילה ביותר את המהות ועיקרי הפעולה של המישק לתכנות יישומים ב-OS/2. בכל המקרים, סגנון התכנות שנשתמש בו מתאר בצורה אופטימלית את ה-API והוא גם מקל מאוד על קריאת התכנית. לדוגמה, רוב המשתנים בדוגמאות מוגדרים כמשתנים גלובליים בהתחלת התכנית, כדי להקל על ההבנה של תפקידם בהמשך. באופן טבעי, בסביבת פיתוח של תוכנה, יש כאלה המתאמצים לצמצם למינימום את מספר המשתנים ולהשאיר רק את אלה שהשגרות צריכות להכנס אליהם. בדוגמאות אלו אין אנו בודקים את ה-ErrorCode המוחזר משגרות הביצוע של OS/2. במספר מקרים הצגנו בצורה מקוצרת עקרונות בסיסיים לתכנות של מספר פעולות בו-זמנית. הקורא המתכוון לפתח תכניות בשפת C ל-OS/2 צריך לפנות לאחד הספרים על C, או לספר הדרכה הדן בפיתוח של יישומים רבי משימות ובטכניקות לפיתוח תכניות ברורות, אשר ניתנות לתחזוקה ולניוד (portable).

בערכת הכלים למתכנת של OS/2 ניתן למצוא מידע חשוב היכול לסייע למתכנת המתעתד לפתח תוכנה בסביבת תפעול זו. בנוסף לקווים המנחים לשימוש במקשר (linker), היא מכילה הגדרות סטנדרטיות של מכללים, הגדרות של פונקציות והודעות שגיאה לשפות אסמבלר/2 ו-C/2, אשר משווקות על-ידי יבמ. כלים אלה, ביחד עם דוגמאות של תכניות המסופקות יחד עם ערכת הכלים, יכולים לעזור למתכנת להפיק תכנית בזמן קצר ביותר.

לפני שנמשיך נסקור בקצרה הידור, קישור וקיצורים מוסכמים בשפת C לתועלת הקוראים, אשר אינם מתמחים בשפה זו דווקא.

התחביר והקיצורים המוסכמים בשפת C

בחלק זה נדון בהידור ובקישור. כמו כן, נסקור חלק קטן של התחביר בשפת C כדי לעזור בקריאת הדוגמאות על-ידי מי שאינם מתכנתים בשפה זו. נדון גם בסוגי הנתונים הנפוצים שמשתמשים בהם כדי לקרוא לפונקציות OS/2 מתוך שפת התכנות IBM C/2.

הידור

תכנית בשפת C מתבססת על קובץ מקור אחד או יותר. הסיומת של שמות קבצי מקור בשפת C היא "c" (xxxxxxxx.c). הכרזות (declarations) והשוואות (equates) נפוצות, הנמצאות בשימוש בתכניות בשפת C, מופרדות בד"כ מהתכנית, ומוכנסות לתוך קבצים הנקראים קבצים מוכללים (include files). הסיומת של שמות הקבצים המוכללים, הידועים גם בשם קבצי כותרת (header files), היא האות "h" (xxxxxxxx.h).

כל קובץ מקור של תכנית חייב לעבור דרך המהדר של שפת C. הפלט של תהליך זה הוא קובץ היעד (object file), שהוא הייצוג בשפת מכונה של תכנית המקור. בנוסף לכך, ניתן למצוא בו מידע המאפשר עבודה עם תכנית הקישור (linker), ומידע למטען (loader) של מערכת ההפעלה על אפשרויות הניוד של התכנית בזיכרון. המהדר יכול גם להפיק קבצי הדפסה (listing files) הערוכים במבנים שונים, שבהם נרשמים התוצאות של תהליך ההידור. אפשר להפעיל את מהדר IBM C/2 מתוך OS/2 על-ידי הפקודה **cc**. לאחר הופעת מנחה המהדר (prompt) יש להכניס את שמות הקבצים הדרושים ואת האופציות השונות. אם תבחר להשתמש בפקודת CC, תרצה בוודאי לבנות קובץ batch (סיומת bat.* ב-DOS ו-cmd.* ב-OS/2), ולהריץ את המהדר ישירות משורת הפקודה. התחביר של הפקודה המאפשרת להריץ את המהדר באופן ישיר:

```
cc sourcename [,[objectname]] [,[listingname]][;]
```

הסבר: cc שם-קובץ-מקור [,[שם-קובץ-יעד]] [,[שם קובץ הדפסה]][;]

באופן כללי, הכוונה היא לכך שאתה חייב לספק את שם קובץ המקור. אם לא תספק את שמות קבצי היעד וההדפסה, המהדר ישתמש בברירות המחדל שלו כדי לקבוע זאת. אופציות שונות של המהדר חייבות להופיע לפני התו נקודה-פסיק (;) שבסוף ההוראה.

דוגמה לשורת פקודה המפעילה את המהדר IBM C/2 היא:

```
cc testprog.c,testprog.obj,/Fs testprog.lst /G2;
```

שורת הפקודה גורמת למהדר לעשות את הדברים הבאים:

- הידור של קובץ המקור הנקרא testprog.c.
- קביעת השם testprog.obj לקובץ היעד.
- יצירת קובץ הדפסה הנקרא testprog.lst.

אופציית Fs/ שלפני שם קובץ ההדפסה מורה לבצע תדפיס (listing) של קובץ המקור. האופציה G2/ מורה להפיק קוד למיקרו-המעבד 80286.

יש להדגיש, כי מתגי האופציות רגישים לשינויים בסוג האות (אות גדולה או קטנה) ויש להכניסם בדיוק כפי שרשום. למרות שהדוגמה הזו מציגה שימוש בשתי אופציות בלבד, אין להסיק מכך ששתי האופציות האלו הן היחידות. קיימות אופציות רבות שניתן להשתמש בהן כדי לשלוט בתהליך ההידור. חלק מהאפשרויות האחרות של המהדר הופכות אותו לכלי חזק מאוד לפיתוח תוכנה בידי המתכנת, אשר יכול לעשות את הדברים הבאים:

- להשתמש במשתני סביבה המכוונים את המהדר לסרוק במדריכים נפרדים אחר קבצים הניתנים לביצוע, קבצים מוכללים ומודולים של ספריות.
- לקבוע אופציות המאפשרות תאימות עם מהדר שפת C גירסה 1.00 למחשבי IBM PC.
- לשלוט על הפקת הסוגים השונים של קבצי ההדפסה.
- לקבוע שיטות שונות לטיפול בחישובים עם נקודה צפה.
- לקבוע את רמת הדיווח של הודעות אזהרה/שגיאה.
- לערוך אופטימיזציות שונות, כמו גודל זיכרון ומהירות ביצוע.
- להשתמש במספר מודלים שונים של זיכרון הכוללים קטן, בינוני, קומפקטי, גדול וענק (Small ,Medium ,Compact ,Large & Huge).
- להפיק קבצי יעד עם המאפיינים הדרושים לעבודה עם תכנית לניפוי שגיאות (debugging) המסופקת יחד עם המהדר.

סיכום קצר זה של האופציות האפשריות במהדר IBM C/2 מכוון להציג מקצת מן הכוח והגמישות שבו. קיימות אופציות נוספות מתקדמות יותר. לשם כך יש לפנות לפרסומים על מהדר זה.

קישור

לאחר שהתכנית עברה הידור, היא מוכנה לקישור. בדומה למהדר, תכנית הקישור (linker) תנחה אותך להכניס את הקלט הדרוש, אם לא סיפקת אותו בשורת הפקודה. לחילופין, תכנית הקישור יכולה לקרוא כקלט "קובץ תגובה" (response file). קובץ זה מפרט את שמות הקבצים ואת האופציות שיהיו בשימוש. למרות שאנו מראים כאן דוגמה לשורת פקודה המפעילה את תכנית הקישור, אנו עוסקים בפרק 7 ביתר פירוט בתחביר של תכנית זו.

חלק חשוב של תהליך הקישור לתכניות בשפת C הוא קביעת אפשרות המיעון לספריות המתאימות של תכניות יעד סטנדרטיות בשפת C. פונקציות רבות הנפוצות בתכניות בשפת C, אינן למעשה חלק מהשפה, אך מקובלות כתקן בהתקנות שונות בשפת C. למשל, ישנן פונקציות סטנדרטיות בשפת C לטיפול בק/פ ובמחרוזות בזמן ריצה. יש ספריות שונות לזמן ריצה המותאמות למודלים שונים של הזיכרון וסביבות שונות של מערכת ההפעלה. אתה יכול לבצע את תכנית הקישור באותה סביבה שבה אתה מתעתד לבצע את התכנית הסופית. במקרה זה השימוש בספריות המתאימות נעשה לפי ברירות המחדל.

בדומה למהדר, נתרכז בהפעלה של תכנית הקישור מתוך שורת פקודה. דוגמה לשורת פקודה המפעילה את תכנית הקישור ב-OS/2 מובאת להלן:

```
link testprog.obj,testprog.exe,testprog.map,
          slibc.lib slibc5.lib doscalls.lib;
```

שורת הפקודה תגרום לתכנית הקישור לעשות את הדברים הבאים:

- קישור של קובץ היעד testprog.obj.
- קביעת השם testprog.exe לקובץ המתקבל. קובץ זה מכיל את התכנית שניתנת כבר לביצוע (executable).
- הכנת מפת קישור testprog.map, המפרטת את כל הסגמנטים בתכנית הטעינה exe.
- שימוש ב-slibc.lib - החלק שאינו תלוי ב-DOS, של הספריות המשרתות תכניות C בזמן ריצה.
- שימוש ב-slibc5.lib - החלק התלוי ב-OS/2, של הספריות המשרתות תכניות C בזמן ריצה.
- שימוש ב-doscalls.lib - הספריה המשמשת להפניות חיצוניות לקישור דינמי חיצוני של פונקציות OS/2.

הפרסומים על IBM C/2 וה-OS/2 Technical Reference מכילים תיעוד על האופציות וברירות המחדל של תכנית הקישור. אנו נמשיך בדיון שלנו על תכנית הקישור בפרק 7.

הידור וקישור בצעד אחד

ב-IBM C/2 יש פקודה המאפשרת למתכנת לבצע הידור וקישור של תכנית בשלב אחד. פקודה זו נקראת פקודת "CL". דוגמה לפקודת CL עבור הידור וקישור של דוגמת התכנית הראשונה בפרק זה, הנקראת name.c, מובאת להלן:

```
cl /Fs name.c /F 2000
```

שורת פקודה זו מכוונת את המהדר ואת תכנית הקישור לעשות את הדברים הבאים:

- האופציה Fs/ מורה להפיק רישום של תכנית המקור.
- "name.c" הוא שמו של קובץ המקור.
- האופציה F 2000/ מקצה מחסנית בת 8KB.

רוב האופציות האפשריות עם פקודת CC אפשריות גם עם הפקודה CL. כפי שניתן לראות, התחביר של שורת הפקודה בפקודה CL הוא מאוד נוח. בהמשך השתמשנו בפקודה CL להידור וקישור של הדוגמאות המוצגות בספר.

קריאה של תכנית OS/2 הכתובה בשפת C

כפי שהוזכר קודם, הכרזות ומשוואות נפוצות, שמשמשות תכניות שונות בשפת C, מופרדות מהתכניות ומוכנסות לתוך קבצים מוכללים. ערכת הכלים למתכנת ב-OS/2 מספקת הכרזות (declarations) נפוצות לפונקציות ב-OS/2 שניתן להשתמש בהן בתכנית בשפת IBM C/2. ניתן לכלול בתכנית את ההכרזות האלו באמצעות משפט התכנות הבא:

```
#include <doscall.h>
```

בתרשים 8 מוצגת דוגמה להכרזה על פונקציה של OS/2 בתכנית בשפת C. אנו משתמשים בפונקציה היפותטית הנקראת "DosGenericCall".

```
/* This is a comment */

/***     DosGenericCall - Read Character from device
 *
 *       Return a character from the device
 */

extern unsigned far pascal DosGenericCall (
        struct DeviceData far *,         /* Buffer for device data */
        unsigned,                        /* Function parameter */
        unsigned );                      /* Device handle */
```

תרשים 8. הכרזה על פונקציה של OS/2 בשפת C
המסופקת בערכת הכלים למתכנת

דבר ראשון שאנו רואים הוא, שהערות מתחילות ב-"*/" ומסתיימות ב-"/*" ויכולות להקיף מספר שורות. הדוגמה מראה הכרזה בתכנית בשפת C על פונקציה חיצונית של OS/2. המונח "unsigned far pascal" מורה על סוג הערך החוזר וגודלו (שהוא קוד השגיאה החוזר מהקריאה לפונקציה), יחד עם הקיצור המוסכם לקריאה לפונקציה. בהמשך הפונקציה ניתן למצוא רשימה של סוגי הפרמטרים עם הערות המסייעות לתעד את סוג הפרמטרים הדרוש לביצוע.

המונח "extern" מפרט את סוג האחסון ויש להשתמש בו בכל הכרזה על פונקציות חיצוניות. זאת אומרת, שעבור קישור פונקציה זו יש להקצות שטח קבוע (static) במשך כל זמן ההרצה של התכנית. השם של שטח זה חייב להיות ידוע לתכנית הקישור.

המונח "unsigned" פירושו, שהערך המוחזר משיגרת הביצוע הוא מספר שלם חיובי, אשר ב-C/2 הוא ערך בן 16 סיביות (מלה של 80286). המונח "far" מורה על כך שצריך להכנס לפונקציה דרך קריאה רחוקה. המונח "pascal" מורה על סדר הכנסת הפרמטרים למחסנית.

רשימת סוגי הפרמטרים נמצאת בתוך הסוגריים של הכרזת הפונקציה. בנוסף לפרמטרים שהם ערכים שלמים וחיוביים, רואים משפט בשפת C המגדיר פרמטר שהוא מחוון רחוק (far pointer) אל מבנה נתונים מסוים. בדיון שלנו על המיקרו-מעבד 80286 הסברנו שהכתובת מורכבת משני חלקים בני 16 סיביות כל אחד. במצב אמיתי הכתובת מורכבת מסגמנט בן 16 סיביות ומכתובת יחסית בת 16 סיביות. במצב מוגן הכתובת מורכבת מערך בורר בן 16 סיביות ומכתובת יחסית בת 16 סיביות. אם התכנית בשפת C פונה לנתונים, היא עושה זאת בעזרת אחד מִשני סוגים של מחוונים. אם הנתון נמצא בסגמנט הנתונים הנוכחי, אז אפשר להשתמש במחוון המפרט את הכתובת היחסית בלבד בסגמנט הנתונים הנוכחי. אם הנתון נמצא בסגמנט נתונים אחר, יש צורך להשתמש במחוון שיפרט הן את ערך הבורר (או הסגמנט במצב אמיתי) והן את הכתובת היחסית.

כדי להשלים את הדוגמה על הכרזה של פונקציה, הבה נסתכל בתרשים 9, המראה הכרזת מבנה ל-DeviceData בפונקציה DosGenericCall.

הכרזות אלו על פונקציה ומבנה הן הצהרות היפותטיות בלבד, אך הן נותנות תמונה ברורה על המבנה של הכרזות סטנדרטיות ב-C/2. הכרזות כאלו קיימות כקבצים מוכללים בערכת הכלים והדבר מקל מאוד על פיתוח תכניות בשפת C בסביבת OS/2.

```
/***    DeviceData - structure that contains device data */

struct DeviceData {
        unsigned char device_code;      /* raw device code */
        unsigned char ascii_code;       /* ASCII mapping of code */
        unsigned long time;             /* time stamp of data */
        };
```

הכרזת המבנה הנראית בתרשים 9 נקראת DeviceData. יש בה 3 פריטים מסוגים שונים. סוגי הנתונים הנפוצים ביותר ב-C/2 לצורך פרמטרים המועברים לפונקציות של OS/2 הם:

char	תו בן 8 סיביות (בתחום 128- עד 127)
unsigned char	תו בן 8 סיביות ללא סימן (מ-0 עד 255)
int	מספר שלם בן 16 סיביות (32,768- עד 32,767)
unsigned	מספר שלם ללא סימן (0 עד 65,535)
long	מספר שלם ארוך בן 32 סיביות (2,147,483,648- עד 2,147,483,647)
unsigned long	מספר שלם ארוך בן 32 סיביות ללא סימן (0 עד 4,294,967,295)
unsigned far *	מחוון בן 32 סיביות (סגמנט/ערך_בורר:כתובת_יחסית) למספר שלם בלי סימן.
char far *	מחוון בן 32 סיביות (סגמנט/ערך_בורר:כתובת_יחסית) לתו.

נבחן עתה מספר דוגמאות פשוטות של השימוש במישק לתכנות יישומים (API).

ק/פ באמצעות תצוגה ומקלדת

הפונקציות למקלדת ולתצוגה ב-OS/2 מהוות תחליף בעל רמת ביצועים גבוהה לפונקציות ה-BIOS הקיימות ב-DOS. מתכנתים יכולים להשתמש בפונקציות אלו כדי לפתח יישומים יעילים לעיבודי תמליל. בחלק זה נבחן דוגמה ליישומים מבוססי תמליל המשתמשים ב-API לתצוגה ולמקלדת.

תרשים 10 מראה את המהלך הלוגי של התכנית הראשונה שבה נשתמש, כדי לתאר מספר קריאות פשוטות לפונקציות OS/2 המשמשות לתצוגה ולמקלדת. תרשים 11 מכיל את התכנית האמיתית.

```
נקה את המסך
הצג "Please Enter Your Name"
קרא מחרוזת תווים מהמקלדת
הצג את התוים שזה עתה נקראו
סוף
```

תרשים 10. הלוגיקה של תכנית הדוגמה הראשונה - שימוש בשגרות VIO ו-KBD

```
/*****************************************************************************************/
/*  Video and Keyboard Sample Program                                                    */
/*****************************************************************************************/

#include <doscall.h>                                 /* OS/2 API declarations        */
#include <string.h>                                  /* C string functions           */

main()                                               /* Start of C main routine      */
{

/* General Variables - used throughout the program ***************************************/

  unsigned  ErrorCode = 0;                           /* Value returned by OS/2 calls */
  int       RowCounter = 0;                          /* Variables used in "for" loop */
  int       ScreenLength = 25;                       /* to clear the display         */

/* KbdStringIn Variables - used to input user's name *************************************/

  char      CharBuffer[32];                          /* Character buffer             */
  struct    KbdStringInLength KbdLength;             /* Length table(from doscalls.h)*/
  unsigned  IOWait;                                  /* Indicate if wait for char    */
  unsigned  KbdHandle;                               /* Reserved word of zeros       */

/* VioWrtTTY Variables - used to clear display and display prompts ***********************/

  char far *TTYCharStr = "\r\n";                     /* Blank line string            */
  int       TTYLength;                               /* Length of TTY string         */
  unsigned  VioHandle;                               /* Reserved word of zeros       */

/* DosExit Variables *********************************************************************/

  unsigned  ActionCode=1;                            /* Exit all threads in process  */
  unsigned  ResultCode=0;                            /* Result saved for DosCWait    */

/* Start of executable program ***********************************************************/

  for(RowCounter =0; RowCounter <= ScreenLength; RowCounter++)   /* Clear the 25 line screen     */
                                                     /* by writing 25 blank lines.   */
     ErrorCode = VIOWRTTTY(TTYCharStr = "\r\n",      /* VioWrtTTY moves the cursor   */
                           TTYLength = strlen(TTYCharStr),   /* as though the output was to  */
                           VioHandle = 0);           /* a teletype device.           */

  ErrorCode = VIOWRTTTY(TTYCharStr = "Please Enter Your Name: ",   /* Output prompt to the user  to*/
                        TTYLength = strlen(TTYCharStr),            /* enter their name.            */
                        VioHandle = 0);

  KbdLength.Length = 32;                             /* Read the user's name into a  */
                                                     /* 32 character keyboard input  */
  ErrorCode = KBDSTRINGIN((char far *)CharBuffer,    /* buffer.                      */
                          (struct KbdStringInLength far *)&KbdLength,   /* The default ASCII keyboard   */
                          IOWait = 0,                /* input mode is used.          */
                          KbdHandle = 0);

  CharBuffer[KbdLength.LengthB] = '\0';              /* Concatenate the user's name  */
  TTYCharStr = strcat("\r\nOS/2 Says Hello To ",CharBuffer);   /* with a "Hello" message.      */

  ErrorCode = VIOWRTTTY(TTYCharStr,                  /* Output the resulting message.*/
                        TTYLength = strlen(TTYCharStr),
                        VioHandle = 0);

  DOSEXIT(ActionCode,                                /* Notify OS/2 of termination   */
          ResultCode);

}                                                    /* End of C main routine        */
/*****************************************************************************************/
```

תרשים 11. תכנית דוגמה - שימוש בשגרות VIO ו-KBD

תיאור של תכנית הדוגמה

התכנית הפשוטה שבתרשים 11 אינה מנצלת את כל הגמישות שמאפשרות הפונקציות לתצוגה ולמקלדת. התכנית מכוונת להראות דרכים נוחות לביצוע קריאות ל-OS/2. נעבור על התכנית שלב אחר שלב, נתאר מה היא עושה, ונסביר באיזה פונקציות של OS/2 היא משתמשת.

לאחר ההערות בתחילת הקובץ הכוללות את שם התכנית, אנו רואים את הפקודה

#include של הקדם-מעבד (precompiler). משפטים אלה מנחים את הקדם-מעבד לכלול הכרזות לספריית ה-API ב-OS/2 ולספריית C שנמצאות בה פונקציות סטנדרטיות לטיפול במחרוזות. שאר התכנית מוגדרת על-ידי פרוצדורה הנקראת "main". בשפת C פרוצדורה (הליך) נקראת "פונקציה". בספר זה נשתמש במונח זה בצורה דומה, אם כי לא בהכרח זהה. בכל תכנית C חייבת להיות פונקציה הנקראת main, מכיון שהמשפט הראשון הניתן לביצוע ב-main מוגדר כנקודת הכניסה הראשית לתכנית C. הגוף של הפונקציה main נמצא בתוך סוגריים ימניים ושמאליים ({}). גוף הפונקציה מתבסס על הכרזה של רשימה של משתנים, ועל סידרה של משפטים הידועים בתחביר של C **כמשפט מורכב** (compound statement).

כדי לעזור בקריאת התכנית, קיבצנו את כל ההכרזות על המשתנים. בקבוצה הראשונה של המשתנים רואים הכרזה על מספר שלם ללא סימן, הנקרא ErrorCode. שפת C מאפשרת לנו לאתחל את המשתנה הזה בהכרזה שלו, ואנו עשינו זאת על-ידי הצבת הערך 0 ב-ErrorCode. הקבוצה הבאה של המשתנים משמשת לקריאה לפונקציה של OS/2 הנקראת KbdStringIn. תחילה אנו רואים מערך של תווים בן 32 בתים הנקרא CharBuffer. אחר-כך אנו רואים משתנה הנקרא KbdLength, שהוא נתון מובנה מסוג KbdStringInLength. המבנה KbdStringInLength מוגדר בקובץ המוכלל "doscalls.h" המסופק יחד עם ערכת הכלים למתכנת ב-OS/2. ההגדרה של מבנה זה מבוססת על ההגדרה של ה-API ל-KbdStringIn כפי שמפורט ב-OS/2 Technical Reference, כרך 2. שני המשתנים הנוספים המשמשים את הפונקציה KbdStringIn הם מספרים שלמים ללא סימן (חיוביים).

הגדרנו שלושה משתנים עבור הפונקציה VioWrtTTY. הראשון, TTYCharStr, הוא מחוון רחוק למחרוזת של תווים. המחוון מצביע בתחילה על שורה ריקה מתווים. ה-r\ וה-n\ הם צירופים של תווים המאפשרים חילוף (escape) לתווי ה-carriage return וה-line feed בהתאמה. TTYLength ו-VioHandle מוגדרים כמספר שלם וכמספר שלם ללא סימן, בהתאמה.

לבסוף, שני המשתנים בפרמטרים הדרושים לפונקציית DosExit מוגדרים כמספרים שלמים ללא סימן ומאופסים בעת האתחול.

הפונקציות של OS/2

לפני שנמשיך, הבה נבחן ביתר פירוט שלוש פונקציות של OS/2 שהצגנו בתכנית המוצגת בתרשים 11.

VioWrtTTY

VioWrtTTY מעבירה לתצוגה מחרוזות של תווים. התצוגה מתחילה במקום הנוכחי של הסמן ואם התווים הגיעו עד סוף השורה, היא עוברת לשורה הבאה. הפונקציה מעלה את המסך שורה אחת למעלה (scroll), אם יש צורך בשורה חדשה בתחתית המסך. נוח מאוד להשתמש בפונקציה אם יש צורך להשתמש בפלט המבוסס על שורות.
לפונקציה זו יש שלושה פרמטרים:

CharStr הוא מחוון רחוק למחרוזת שצריכה להכתב.

Length הוא מספר שלם המציין את אורך המחרוזת בפלט.

VioHandle המקשר לתצוגה אינו נמצא כרגע בשימוש במישק לתכנות התצוגה, אבל המתכנת חייב לספק מלה של אפסים לפרמטר זה.

כל התווים הנמצאים במחרוזת הפלט נשלחים לתצוגה פרט ל-carriage_return, line_feed, backspace, tab ותווים המפעילים את הזמזם. לתווים אלה מתייחסים כפקודות ופועלים לפיהן כנדרש.

KbdStringIn

פונקציה זו קוראת מחרוזת תווים מהמקלדת. התנהגות הפונקציה מושפעת ממצב העבודה של המקלדת. תיאור מפורט של מצבי העבודה של המקלדת הוא מחוץ להיקף של ספר זה. למרות זאת, ברצוננו לציין מספר מצבים: ASCII מול בינארי, ומצב Echo On/Off.
לפונקציה זו יש ארבעה פרמטרים:

CharBuffer הוא מצביע רחוק למאגר (buffer) של מחרוזת הקלט.

length הוא מצביע רחוק לטבלת האורכים (מבנה) עם שני איברים. האיבר הראשון בטבלה מועבר למשתמש בפונקציה זו ומכיל את האורך של מאגר הקלט. האיבר השני הוא האורך של הקלט המתקבל מהפונקציה KbdStringIn.

IOWait הוא פרמטר בעל מספר משמעויות התלויות במצב העבודה של המקלדת. במצב ASCII, פרמטר זה תמיד נקבע ל-0 כדי להצביע על כך שהפונקציה צריכה לחכות שהמשתמש ילחץ על return. במצב בינארי, ערך "אפס" בפרמטר זה מצביע על כך שהפונקציה צריכה לחכות עד אשר מאגר הקלט מלא, ורק אז היא חוזרת. ערך "אחד" בפרמטר זה במצב בינארי משמעותו לחזור מיד, עם כמה שיותר תווים שיכולים להכנס למאגר הקלט.

KbdHandle מציין את ברירת המחדל של המקלדת, או את המקלדת הלוגית. יישום יכול לעבוד עם מספר מקלדות לוגיות כחלק מהתמיכה של OS/2 בדפי קוד (Code Pages) (ראה פרק 8 בדיון על דפי קוד). בדוגמה שלנו בחרנו את מצב ברירת המחדל למקלדת על-ידי כך שקבענו ש-KbdHandle יהיה שווה לאפס.

בדוגמה שלפנינו בחרנו להשתמש במצב ברירת המחדל למקלדת: ASCII עם Echo On.

DosExit

פונקציה זו מודיעה למערכת ההפעלה שמופסקת הפעולה של thread, או תהליך.
לפונקציה זו יש 2 פרמטרים:

ActionCode מציין אם להפסיק את פעולתו של thread זה בלבד (ActionCode = 0), או להפסיק את פעולתם של כל ה-threads בתהליך (ActionCode = 1).

ResultCode הוא קוד הפסקת הפעולה המועבר לכל thread בתהליך האב שהנפיק את הפקודה DosCWait (המתן שתהליך הבן יסיים את פעולתו) לתהליך זה. במקרה של תהליך בן עם threads רבים, רק קוד הפסקת הפעולה האחרון יוחזר ל-thread הממתין עם DosCWait.

כאשר תהליך עומד להסתיים, מבצע ה-thread האחרון בו את השגרות הנמצאות ברשימת DosExitList. כאשר תהליך מסיים את פעולתו, משוחררים כל המשאבים, שנערך אחריהם מעקב של המערכת והם היו בבעלות התהליך. הפסקת פעולתו של תהליך חייבת להיות מצוינת תמיד על-ידי ActionCode = 1, כי המערכת יכולה ליצור בצורה אוטומטית threads בשמו של התהליך לצורך מספר פעילויות שונות.

המשך התיאור של תכנית הדוגמה

נתחיל לבצע את תכנית התצוגה והמדפסת בפעולה של ניקוי המסך. אנו משתמשים ב-VioWrtTTY בתוך לולאה מסוג "for", כדי להוציא פלט של 25 שורות ריקות למסך. מאחר ותכניות C רגישות לצורת האותיות (גדולות או קטנות באנגלית), כל פונקציות ה-OS/2 נכתבו באותיות גדולות בלבד. אנו נמשיך להשתמש בצורה מעורבת בזמן הדיון על הפונקציות, כדי לשפר את הקריאות של התכנית. הביטוי "RowCounter++" בלולאת ה-for פירושו הוא "השתמש בערך הנמצא ב-RowCounter ולאחר מכן העלה אותו באחד". שים לב שאורך המחרוזת בפלט מתקבל על-ידי שימוש בפונקציית אורך מחרוזת סטנדרטית של שפת C (strlen).

הצעד הבא הוא להוציא מנחה (prompt) למשתמש כדי שיוכל להזין את שמו/ה. כהכנה לקריאה לפונקציה KbdStringIn, קבענו ערך תחילי לאיבר המכיל את גודל מאגר הקלט. רק לאחר מכן קראנו ל-KbdStringIn.

שים לב שלא הגדרנו מחוון ל-buffer שיועבר כפרמטר לפונקציה זו. במקום זאת, נתנו תפקיד לביטוי זה ברשימת הביטויים של הפונקציה KbdStringIn. ב-C, הערך של מזהה המערך הוא אקוויוולנטי למחוון של האיבר הראשון (אינדקס המערך שווה לאפס) במערך. מכיון שאנו מהדרים את הדוגמה כתבנית של תכנית קטנה, ברירת המחדל של המחוונים לנתונים היא ערך בן 16 סיביות של הכתובת היחסית בתוך הסגמנט. הפונקציה KbdStringIn דורשת מחוון רחוק למאגר של מחרוזת הקלט. עשינו זאת בעזרת הביטוי "(* char far)".

באופן דומה, המחוון לטבלת האורכים (מבנה) של מחרוזת הקלט מוכל בפרמטר השני. אפשר לקרוא כך את הביטוי בתכנית: "מחוון רחוק למבנה מסוג KbdStringInlength הנמצא בכתובת KbdLength". הפלט הסופי של התכנית הפשוטה הזו נבנה על-ידי הצבה של בית ריק במאגר הקלט בסוף המחרוזת שהמשתמש הכניס. בית ריק זה הופך את המחרוזת למחרוזת ASCIIZ שניתן להעביר אותה לפונקציה המבצעת שרשור (strcat). שם המשתמש משורשר אז עם הודעת ה-hello, והמחרוזת הסופית מועברת לתצוגה בעזרת הפונקציה VioWrtTTY.

השלב האחרון של התכנית הוא לקרוא ל-DosExit, כדי להודיע למערכת שהתכנית עומדת לסיים את פעולתה. שים לב, כי הערך התחילי של ה-ActionCode הוא 1, כדי להודיע למערכת שתפסיק את הפעולה של כל thread שהיא התחילה בשמו של התהליך.

בתכנית זו ניתן לראות כמה קל להריץ תכנית תחת OS/2. לאחר שהבנו את הדוגמה הזאת, נעבור לדוגמה שתנצל כמה מהמאפיינים החדשים של OS/2.

ביצוע רב-משימות - תהליך אחד עם שני THREADS

כפי שראית בפרק 3, OS/2 מספקת אוסף גדול ומקיף של פונקציות המאפשרות ביצוע רב-משימות. נסתכל על תכנית פשוטה המתארת את השימוש בתהליך אחד עם שני threads, המנצלים את מנגנון התקשורת בין תהליכים (InterProcess Communications - IPC), כדי לסנכרן את עצמם.

אחד המאפיינים החשובים של ה-threads הוא, שהם קשורים בצורה הדוקה (tightly coupled). אנו מתכוונים בכך שלכל ה-threads בתוך אותו תהליך יש אפשרות שווה לגשת למשאבים הנמצאים בבעלות התהליך. הזיכרון המוקצה לתהליך הינו אחד מהמשאבים המשותפים ל-threads. כדי לנצל אותו כראוי, אנו משתמשים באתת RAM כמנגנון IPC המסנכרן את ה-threads.

התכנית שלנו הינה דוגמה ליישום המשתמש ב-thread אחד לביצוע פרוצדורה המקבלת נתונים מהמקלדת, וב-thread אחר - כדי לעדכן את המסך. תרשים 12 מציג את הלוגיקה של התכנית הזו.

שים לב שלא סיימנו במפורש את פעולתו של thread מס' 2, מכיון שקבענו שה-ActionCode בפונקציה DosExit יהיה שווה ל-1. בכך מודיעים למערכת לסיים את התהליך וגורמים לסיום כל ה-threads השייכים לתהליך, ובתוכם - thread מס' 2.

תרשימים 13 ו-14 הם דוגמת התכנית עם threads רבים.

thread מס' 1	thread מס' 2
פרוצדורה ראשית ▪ קבע את האתת שיצביע מתי לחכות לתו ▪ התחל את thread מס' 2 ובצע את פרוצדורת המקלדת ▪ קרא לפרוצדורת התצוגה ▪ סוף תהליך פרוצדורת התצוגה ▪ בצע עד אשר המשתמש יכניס את התו "q" ▪ המתן עד אשר האתת יהיה נקי ואז קבע אותו מחדש ▪ הצג הודעה שמקש נלחץ ▪ חזור לפרוצדורה הראשית	פרוצדורת המקלדת ▪ ביצוע אינסופי ▪ המתן לקריאת תו ▪ נקה את האתת

תרשים 12. הלוגיקה של התכנית בעלת שני threads

```
/*********************************************************************************************/
/*  Process with 2 Threads Sample Program                                                    */
/*********************************************************************************************/

#include <doscall.h>                                        /* OS/2 API dynamic link library*/
#include <stdio.h>                                          /* C standard I/O run time lib  */

/* General Variables *************************************************************************/

  unsigned            ErrorCode = 0;                        /* Error code return from OS/2  */
                                                            /* function calls               */
/* Semaphore Function Variables **************************************************************/

  unsigned long       RamSemaphore;                         /* Storage for RAM semaphore    */
  unsigned long far   *SemHandle = &RamSemaphore;           /* Pointer to RAM semaphore     */
  unsigned long       Timeout = -1;                         /* Set for infinite timeout     */

/* KbdCharIn Variables ***********************************************************************/

  struct KeyData      KeyStructure;                         /* Key structure defined in     */
                                                            /* doscalls.h                   */
  unsigned            IOWait;                               /* Indicate if wait for char    */
  unsigned            KbdHandle;                            /* Reserved word of zeros       */

  struct KeyData far *KeyStructurePointer = &KeyStructure;  /* Pointer to key structure     */

/* DosExit Variables *************************************************************************/

  unsigned            ActionCode=1;                         /* Exit all threads in process  */
  unsigned            ResultCode=0;                         /* Result saved for DosCWait    */

/*********************************************************************************************/

  void                DisplayProcedure();                   /* Function decarations         */
  void far            KeyboardProcedure();
```

תרשים 13. threads רבים (חלק 1) - הכרזות

הפונקציות של OS/2

אנו מניחים בנקודה זו שאין צורך לחזור על הכרזות המשתנים בשאר תכניות הדוגמה, ולכן ניגש מיד להגדרות של הפונקציות של OS/2. בדוגמת התכנית שבתרשימים 12 עד 14 יש חמש פונקציות חדשות של OS/2.

DosSemSet

פונקציה זו קובעת אתת ויש לה פרמטר אחד.

SemHandle הוא ערך בגודל מלה כפולה שהמשמעות שלו תלויה בסוג האתת. לגבי אתת מערכת, המקשר מוחזר הן בבקשת DosCreateSem והן בבקשת DosOpenSem. לגבי אתת ה-RAM, המקשר הוא הכתובת של האתת בזיכרון (מחוון רחוק לאתת).

הפונקציה קובעת את האתת בלי להתייחס למצבו הקודם. בתכנית הדוגמה השתמשנו במאפיין זה כדי לאתחל את האתת.

DosSemClear

DosSemClear מנקה את האתת ומעורר כל thread שהמתין לו. ישנו רק פרמטר אחד לפונקציה זו.

SemHandle (כמו בפונקציה DosSemSet)

```
main()                                                              /* Start of main procedure      */
{

/* DosCreateThread Variables *****************************************************************************/

  unsigned          ThreadIDWord;                                   /* New thread ID                */
  unsigned char     NewThreadStack[2000];                           /* New thread stack             */

/* Start of executable program **************************************************************************/

  ErrorCode = DOSSEMSET(SemHandle);                                 /* Initialize semaphore as set. */

  ErrorCode = DOSCREATETHREAD(KeyboardProcedure,                    /* Create thread 2 for keyboard */
                          (unsigned far *)&ThreadIDWord,            /* procedure.                   */
                          (unsigned char far *)&NewThreadStack[1998]);

  printf("Created separate thread (thread 2) for the Keyboard procedure.\n"  /* Display status message */
        "Initial thread (thread 1) is executing the Display procedure.\n"
        "Press any alphanumeric key, press q to quit.\n");

  DisplayProcedure();                                               /* Call the display procedure   */
                                                                    /* with thread 1.               */
  printf("Issuing DosExit with ActionCode set to terminate all threads.");  /* Display the ending status */
                                                                    /* message.                     */
  DOSEXIT(ActionCode,                                               /* Notify OS/2 of termination   */
          ResultCode);

}                                                                   /* End of C main routine.       */
/*******************************************************************************************************/

void DisplayProcedure()                                             /* Display procedure.           */
{
  do{
    ErrorCode = DOSSEMREQUEST(SemHandle,                            /* Block on the SemRequest until*/
                              Timeout);                             /* a key press causes the       */
    printf("%c key was pressed, press q to quit.\n",KeyStructure.char_code);  /* Keyboard procedure to issue */
  }while(KeyStructure.char_code != 0x71);                           /* a SemClear.                  */
  return;
}                                                                   /* End of Display procedure     */
/*******************************************************************************************************/

void far KeyboardProcedure()                                        /* Keyboard procedure.          */
{
  for(;;){                                                          /* Infinite loop blocks waiting */
    ErrorCode = KBDCHARIN(KeyStructurePointer,                      /* for a character and then     */
                          IOWait = 0,
                          KbdHandle = 0);
    ErrorCode = DOSSEMCLEAR(SemHandle);                             /* signals the Display procedure*/
  }                                                                 /* via SemClear                 */
}                                                                   /* End of Keyboard procedure.   */
/*******************************************************************************************************/
```

תרשים 14. threads רבים (חלק 2) - פרוצדורת ה-main

הפונקציה מנקה את האתת בלי להתייחס למצבו הקודם. ניתן להשתמש בה עם הפונקציות DosSemRequest, DosSemSetWait, DosSemWait ו-DosMuxSemWait כדי לתמוך הן בשיתוף משאבים והן באיתות.

DosSemRequest

פונקציה זו משיגה את הבעלות על האתת. הכוונה בכך היא, שהפונקציה משנה בהצלחה את מצב האתת מלא-פעיל לפעיל. בדוגמת התכנית שלנו, השתמשנו ב-DosSemRequest עם אתת לא-בלעדי, כדי לאותת על אירועים. אם האתת פעיל בזמן שפונקציה זו נכנסת אליו, ה-thread יעבור למצב המתנה עד אשר האתת יתנקה. ה-thread המבקש יכול לציין פסק זמן שלאחריו יופסק מצב ההמתנה וה-thread יוחזר למשתמש בפונקציה זו.
לפונקציה זו יש שני פרמטרים.

SemHandle (כמו בפונקציה DosSemSet)

Timeout הוא ערך בגודל של מלה כפולה המצביע על מספר אלפיות השניה שה-thread יחכה לאתת. המשתמש בפונקציה זו יכול לוזורות על פסק זמן שיימשך עד אין סוף על-ידי הצבה של (1-) בפרמטר.

אפשר להשתמש בפונקציה זו עם אתתי מערכת, כדי להסדיר שימוש סידרתי במשאב, אשר במסגרתו ניתן להגדיר אתת מערכת עם בעלות בלעדית. זאת אומרת, שברגע ש-thread מסוים קבע את בעלותו על האתת, thread אחר לא יוכל לנקות, או לשנות את המצב שלו. מערכת ההפעלה עורכת מעקב אחר מספר הפעמים שנעשה שימוש ב-DosSemRequest. האתת ישתחרר רק לאחר שהפונקציה DosSemClear תבוצע מספר פעמים זהה. דיון מפורט על השימוש באתתים בלעדיים מופיע בחלק על האתתים בפרק 3.

DosCreateThread

פונקציה זו יוצרת thread נפרד בתוך התהליך הנוכחי. יש לה שלושה פרמטרים.

PgmAdress הוא מחוון רחוק לכתובת שתקבל את השליטה במסגרת ה-thread החדש.

ThreadIDWord הוא מחוון רחוק לכתובת של המלה שבה DosCreateThread יציב את המספר המזהה של ה-thread החדש. אפשר להשתמש במספר זה כדי לשלוט על ה-thread עם פונקציות אחרות (כמו הפונקציה DosSuspendThread, או DosResumeThread).

NewThreadStack הוא מחוון רחוק לכתובת שבה נמצא סוף המחסנית של ה-thread החדש. המתכנת אחראי להקצות שטח למחסנית לכל thread חדש של התהליך. הגודל המינימלי של המחסנית תלוי בפונקציות של OS/2 שה-thread יבצע, אך אנו מציעים שלא יהיה קטן מ-2KB.

המערכת יוצרת יישות חדשה (thread) הניתנת לביצוע, ועורכת קריאה רחוקה לתכנית הנמצאת בכתובת המצוינת בפרמטר PgmAddress. כל המשאבים של התהליך משותפים בו-זמנית לשימוש thread זה ולכל שאר ה-threads בתהליך.

KbdCharIn

הפונקציה KbdCharIn מחזירה קבוצה של נתונים כאשר היא מקבלת תו מהמקלדת. ההתייחסות לקבוצת נתונים זו היא כאל רשומה של נתוני תו. לפונקציה זו יש שלושה פרמטרים.

CharData הוא מחוון רחוק למאגר, שבו יוצבו נתוני התו. קבוצת הנתונים המוחזרת עם כל קבלת תו היא גדולה, כפי שניתן לראות ברשימת השדות הבאה:

- קוד ASCII של התו.
- קוד הסריקה (scan code) מהמקלדת של המקש שהוקש.
- מידע על המצב של התו.
- שדה שמור לצורך מידע על המצב של "התמיכה בשפות לאומיות".
- שדה המתאר את מצב ה-Shift של כל מקשי ה-Shift.
- חותמת הזמן של ההקשה (time stamp of the keystroke).

IOWait מציין אם לחכות לתו (IOWait = 0), או לחזור מיד אם לא ניתן לקבל תו (IOWait = 1).

KbdHandle מציין אם משתמשים בברירת המחדל של המקלדת, או במקלדת לוגית.

רשומת נתוני תו מתקבלת בכל פעם שמתקבל תו, או בכל פעם שמשתנה מצב ההיסט במקלדת. המידע על ההקשה ניתן לשינוי על-ידי תכנית הפיקוח של המקלדת לפני שהנתונים מגיעים ליישום.

תכנית דוגמה בעלת THREADS רבים

בתכנית זו אנו רואים כמה קל ליצור thread עצמאי ביישום של OS/2. אנו משתמשים באתת RAM כדי לסנכרן את הפעולה של שני ה-thread, ומשום כך צריך היה להפעיל את האתת לפני שהתחלנו את ה-thread השני. רק אחר-כך קראנו לפונקציה DosCreateThread. שים לב שלמרות שזו תכנית קטנה בשפת C, אנו צריכים לציין רק את המזהה עבור KeyBoardProcedure. זאת, מאחר שהכרזנו עליה כפרוצדורה רחוקה (void far KeyBoardProcedure).

לאחר החזרה מ-DosCreateThread, אנו משתמשים בפונקציית הפלט הסטנדרטית בשפת C, "printf", כדי להציג הודעה על הסטטוס. לאחר מכן קראנו ל-DisplayProcedure, אשר עוברת מיד למצב המתנה מאחר והיא מבקשת להשתמש באתת שהופעל קודם. לאחר שהאתת נוקה, מדפיסה הפונקציה printf הודעה שמקש נלחץ. תנאי מסוג "while" הופעל, כדי לבחון אם התו שנלחץ היה "q" (ערך הקסדצימלי '71'). אם התו אינו q, הפרוצדורה חוזרת על עצמה ומוציאה שוב DosSemRequest. בכל המקרים האחרים אנו חוזרים לפרוצדורה הראשית, מציגים הודעת סיום, ומפסיקים את פעולתו של התהליך.

פרוצדורת המקלדת היא פונקציה פשוטה מאוד המכילה לולאה אין-סופית מסוג "for" עם שתי קריאות לפונקציות של OS/2. הקריאה הראשונה היא לפונקציה KbdCharIn עם הפרמטר IOWait, הקובע שיש להמתין לתו שייכנס. כאשר התו מגיע, KbdCharIn מסתיימת ומתבצעת הפונקציה DosSemClear, כדי לאותת לפרוצדורת התצוגה (DisplayProcedure).

ביצוע רב-משימות - שני תהליכים בעלי THREADS רבים

כפי שרואים מהדוגמה הקודמת, אפשר להשתמש במספר threads בו-זמנית בתוך יישום אחד. ניתן להשתמש במספר threads כדי לפשט את מבנה התכנית, על-ידי חלוקה של פונקציות היישום לפרוצדורות המתבצעות בצורה נפרדת. חשוב לזכור, ש-threads השייכים לאותו תהליך, קשורים בצורה הדוקה. מאפיין זה יכול לפשט את השימוש ב-threads מחד, אך יכול לפגוע בתכנון המודולרי של התכנית מאידך. כל ה-threads בתהליך יכולים להשתמש במשאבי התהליך בצורה שווה, ולכן כל תקלה ב-thread יכולה להרוס נתונים, או להפריע לביצוע של thread אחר באותו תהליך.

עקרון בסיסי של תכנון מבני הוא צמצום של היקף הנתונים ושל הבעלות על משאבים העומדים לרשות פונקציות עצמאיות. המטרה של עקרון זה היא להפחית את המורכבות של יישום גדול על-ידי פירוקו למערכת של מודולים עצמאיים

קטנים יותר, וכך לקבל יישום קל יותר לפיתוח ולתחזוקה. אחת הטכניקות שמתכנת היישומים ב-OS/2 יכול להשתמש בה כדי לצמצם את היקף הנתונים והבעלות על משאב. היא השימוש בתהליכים של OS/2.

ניקח את הדוגמה הקודמת כבסיס לדוגמה הזו, ונשפר את היישום ההיפותטי שהיא מייצגת. נניח שאנו רוצים להפעיל בסיס נתונים שיכיל נתונים על התרחשות של אירועים שונים. בדוגמה הקודמת, האירוע היחיד שהיינו צריכים לפקח עליו היה לחיצת המקש על-ידי המשתמש. נניח, שאירוע זה הוא רק אחד ממספר סוגים של אירועים שאנו רוצים לרשום בבסיס הנתונים. יתר על כן, גם נניח ששמירה על השלמות של בסיס הנתונים הזה חיונית לעסק שלנו.

הגדרנו כרגע יישום שניתן לחלק אותו בצורה לוגית לשני מודולים עצמאיים. מודול אחד לפיקוח האירועים (זכור, ביישום אמיתי יהיו בוודאי יותר מאשר מודול אחד לפיקוח האירועים) ומודול נפרד אחר - לרישום האירוע בבסיס הנתונים. היישום שלנו מיישם את גישת המודולים בשני תהליכים ומשתמש במנגנון לתקשורת בין תהליכים IPC כדי לקשר ביניהם. כאן, מנגנון IPC מדגים העברת נתונים במסרים (messages) באמצעות מנגנון הצינור.

על-ידי הפרדת הפיקוח על האירועים מהרישום שלהם, הפכנו את היישום לפשוט יותר, כי הבעיה מוצגת עתה כשתי בעיות קטנות ופשוטות יותר (גישת הפרד ומשול). הביצוע של יישום זה בסביבה רבת משימות של OS/2 הוא יתרון, כי היישומים יכולים להתבצע במקביל. השמירה על שלמות הנתונים בבסיס הנתונים משתפרת, כי התהליך המפקח על האירועים אינו יכול לגשת לנתונים ולמשאבים (קבצים) השייכים לתהליך הרישום בבסיס הנתונים. הפעולה ההדדית בין שני התהליכים מוגבלת למנגנון IPC המוגדר היטב - הצינור. הדבר מדגים כיצד OS/2 מפשטת את היישום של עקרונות התכנות המובנה. ניתן לתכנן יישומים בצורה מובנית, עם קישור מינימלי בין מודולים, על-ידי שימוש במישק IPC המוגדר היטב. ולבסוף, הדוגמה שלנו, המשתמשת בתהליכים נפרדים, מדגימה שאם thread השייך לתהליך המפקח על האירועים מפסיק את פעולתו בצורה לא נורמלית, אין אנו צריכים לחשוש מכך שתהליך הרישום בבסיס הנתונים יפסיק גם הוא את פעולתו בצורה לא נורמלית.

ישנה נקודה נוספת לגבי הדוגמה ההיפותטית שלנו. כדי לפשט את הדוגמה ולתת לה יותר מאפיינים הנראים לעין, ערכנו הדמייה של כתיבת רשומות לקובץ של בסיס הנתונים, על-ידי כתיבתן לחלון במסך התצוגה.

הלוגיקה של תכנית הדוגמה הזו מוצגת בתרשים 15. תרשימים 16 עד 25 מכילים את תכניות המקור של הדוגמה העוסקת ביישום עם מספר תהליכים.

הפונקציות של OS/2

בדוגמה המוצגת בתרשימים 15 עד 25 השתמשנו בשמונה פונקציות חדשות של OS/2.

VioScrollUp

פונקציה זו מגוללת כלפי מעלה שטח נתון במאגר התצוגה. לפונקציה זו יש שבעה פרמטרים.

תהליך 1

thread מס' 1

פרוצדורה ראשית

- אתחל את חלונות התצוגה לתהליך זה
- הגדר את הצינור להעברת הנתונים
- התחל תהליך 2 (DosExecPgm)
- קבע והפעל את האתת המצביע שיש לחכות לתו
- הגדר thread 2 כדי שיבצע את פרוצדורת המקלדת
- קרא לפרוצדורת התצוגה
- סיים את התהליך

פרוצדורת התצוגה

- בצע עד אשר המשתמש מכניס את התו "q"
  - המתן עד אשר האתת משתחרר, ואז קבע אותו מחדש
  - קבע מאפיין חדש לתו זה
  - הצג את התו/מאפיין
  - כתוב הודעה אל הצינור
- חזור לפרוצדורה הראשית

thread מס' 2

פרוצדורת המקלדת

- בצע עד אינסוף
  - המתן לקבלת תו
  - הצג את התו וסמן את המאפיין שלו
  - נקה את האתת

תהליך 2

thread מס' 1

פרוצדורה ראשית

- אתחל את חלון התצוגה לתהליך זה
- בצע עד שתתקבל הודעה עם התו "q"
  - קרא הודעה מהצינור
  - שרשר הודעה עם תאריך וזמן
  - כתוב לבסיס הנתונים (הצג)
- סיים את התהליך

thread מס' 2

פרוצדורת תאריך וזמן

- קרא את התאריך והזמן
- בנה מחרוזות של תאריך וזמן
- הצג את מחרוזות התאריך והזמן
- השהה ביצוע למשך שניה אחת

תרשים 15. הלוגיקה של תכנית הדוגמה - שני תהליכים בעלי threads רבים

TopRow ערך בגודל מלה המכיל את השורה העליונה בשטח שיש לגולל.

LeftCol ערך בגודל מלה המכיל את הטור השמאלי ביותר בשטח שיש לגולל.

BotRow ערך בגודל מלה המכיל את השורה התחתונה בשטח שיש לגולל.

RightCol ערך בגודל מלה המכיל את הטור הימני ביותר בשטח שיש לגולל.

Lines ערך בגודל מלה המכיל את מספר השורות שיש לגולל. הוא מצביע על מספר השורות שיוכנסו בתחתית השטח שנגלל.

Cell הוא מצביע רחוק לכתובת של מבנה המכיל את התו ואת המאפיין שלו (attribute), שבהם יש להשתמש בשטח שגולל.

VioHandle נשמר לשימוש עתידי וחייב להכיל מלה של אפסים.

```
/*****************************************************************************************************/
/*  Multiple Processes with Multiple Threads - Process 1                                            */
/*****************************************************************************************************/

#include <doscall.h>                                           /* OS/2 API dynamic link library*/
#include <stdio.h>                                             /* C standard I/O run time lib  */
#include <string.h>                                            /* C string library             */

/* General Variables *********************************************************************************/

  unsigned            ErrorCode = 0;                           /* Error code return from OS/2  */
                                                               /* function calls               */
/* Semaphore Function Variables **********************************************************************/

  unsigned long       RamSemaphore;                            /* Storage for RAM semaphore    */
  unsigned long far  *SemHandle = &RamSemaphore;               /* Pointer to RAM semaphore     */
  unsigned long       Timeout = -1;                            /* Set for infinite timeout     */

/* KbdCharIn Variables *******************************************************************************/

  struct KeyData      KeyStructure;                            /* Key structure defined in     */
                                                               /* doscalls.h                   */
  struct KeyData far *KeyStructurePointer = &KeyStructure;     /* Pointer to key structure     */

/* VioScrollUp Variables *****************************************************************************/

  unsigned            TopRow;                                  /* Upper left hand corner       */
  unsigned            LeftCol;
  unsigned            BotRow;                                  /* Bottom right hand corner     */
  unsigned            RightCol;
  unsigned            NumLines;                                /* Number of lines to scroll    */
  char                FillChar[2] = {0x20,0x1F};               /* Fill character to use        */
  unsigned            VioHandle = 0;                           /* Reserved word of zeros       */

  char                NewFillChar[2] = {0x00,0x0F};            /* Another fill character       */

/* VioWrtCharStrAtt Variables - used to display text prompts and messages ****************************/

  char far           *CharStr;                                 /* String to be written         */
  unsigned            VioLength;                               /* Length of character string   */
  unsigned            Row;                                     /* Starting position - row      */
  unsigned            Column;                                  /* Starting position - column   */
  char far           *Attribute;                               /* Display attribute            */

/* DosExit Variables *********************************************************************************/

  unsigned            ActionCode=1;                            /* Exit all threads in process  */
  unsigned            ResultCode=0;                            /* Result saved for DosCWait    */
```

תרשים 16. תהליכים רבים: תהליך 1 (חלק 1) - הכרזות

מספרי השורה והטור מבוססים על אפס (zero-based).

בנוסף לגלילה (scroll), פונקציה זו יכולה לשמש מנגנון מהיר ונוח למחיקת המסך.

VioWrtCharStr ו-VioWrtCharStrAtt

פונקציות אלו כותבות מחרוזת תווים למסך. עם VioWrtCharStr, התווים במחרוזת מקבלים את מאפייני התצוגה של התווים שהם מחליפים. ל-VioWrtCharStrAtt יש פרמטר נוסף, המאפשר למתכנת לציין מאפיין תצוגה יחיד שישמש להצגת התו במסך.

לפונקציות אלו יש חמישה ושישה פרמטרים, בהתאמה.

CharStr הוא מחוון רחוק למחרוזת תווים שצריכה להכתב למסך.

Length הוא ערך בגודל מלה, המכיל את האורך בבתים של מחרוזת התווים.

```
/* Display window character definitions *******************************************************************/

  char far *Thread1Window[12] = {"┌─┐",                                  /* An array of strings used to  */
                                 "│ │",                                  /* outline the Thread 1 display */
                                 "│ │",                                  /* window                       */
                                 "│ │",
                                 "│ │",
                                 "│ │",
                                 "│ │",
                                 "│ │",
                                 "│ │",
                                 "│ │",
                                 "│ │",
                                 "└─┘"};

  char far *Thread2Window[3] = {"┌─┐",                                   /* An array of strings used to  */
                                "│ │",                                   /* outline the Thread 2 display */
                                "└─┘"};                                 /* window                       */

  int                   RowCounter = 0;                                 /* Variables used in "for" loop */
  int                   Window1Length = 12;                             /* to output arrays of strings  */
  int                   Window2Length = 3;

/* DosExecPgm Variables *************************************************************************************/

  char                  ObjNameBuf[64];                                 /* Object name buffer           */
  char                  ArgumentString[64];                             /* Argument string buffer       */
  struct ResultCodes    ReturnCodes;                                    /* DosExecPgm ResultCode struct */
  char                  ProgramName[] = "proc2.exe";                    /* Program name string          */

  char far              *ObjNameBufPointer = ObjNameBuf;                /* Object name buffer pointer   */
  unsigned              ObjNameLength = 64;                             /* Object name buffer length    */
  unsigned              ExecFlags = 1;                                  /* Execute asynchronously       */
  char far              *ArgPointer = ArgumentString;                   /* Argument string pointer      */
  char far              *EnvPointer = 0x0000;                           /* Environment strings pointer  */
  struct ResultCodes far *ReturnCodesAddress = &ReturnCodes;            /* Return code structure pointer*/
  char far              *PgmPointer = ProgramName;                      /* Program to exec filename ptr */

/* DosMakePipe Variables ************************************************************************************/

  unsigned              ReadHandle;                                     /* Pipe read handle - returned  */
  unsigned              WriteHandle;                                    /* Pipe write handle - returned */
  unsigned              PipeSize;                                       /* Requested pipe size          */

/* DosWrite Variables ***************************************************************************************/

  unsigned              BufferLength = 2;                               /* Write output buffer length   */
  unsigned              BytesWritten;                                   /* Bytes written - Returned     */

/* String variables - used for general string manipulation *************************************************/

  int                   StringIndex;
  int                   Arg1Index;
  int                   Radix = 10;
  char                  HandleString[8];
  char                  *HandleStringPointer;

/************************************************************************************************************/

  void                  DisplayProcedure();                             /* Function declarations        */
  void far              KeyboardProcedure();
```

תרשים 17. תהליכים רבים: תהליך 1 (חלק 2) - הכרזות

Row הוא ערך בגודל מלה, המכיל את מספר השורה (zero-based) שבה יכתב התו הראשון במחרוזת.

Column ערך בגודל מלה, המכיל את מספר הטור (מבוסס-אפס) שבו יכתב התו הראשון במחרוזת.

attribute (רק ב-VioWrtCharStrAtt) הוא מחוון רחוק למאפיין התצוגה, שיש להשתמש בו עם כל תו במחרוזת.

VioHandle שמור לשימוש עתידי, וחייב להכיל מלה של אפסים.

```
main()                                                            /* Start of main procedure      */
{

/* DosCreateThread Variables *****************************************************************************/

  unsigned          ThreadIDWord;                                 /* New thread ID                */
  unsigned char     NewThreadStack[100];                          /* New thread stack             */

/* Start of executable program ***************************************************************************/

  ErrorCode = VIOSCROLLUP(TopRow=0,                               /* Clear the screen             */
                          LeftCol=0,
                          BotRow=-1,
                          RightCol=-1,
                          NumLines=-1,
                          (char far *)FillChar,
                          VioHandle = 0);

  ErrorCode = VIOWRTCHARSTR(CharStr = "OS/2 (TM) Standard Edition",   /* Title the display       */
                            VioLength = strlen(CharStr),
                            Row = 1,
                            Column = 27,
                            VioHandle = 0);

  ErrorCode = VIOWRTCHARSTR(CharStr = "Multiple Processes/Threads Example",  /* Include the program name */
                            VioLength = strlen(CharStr),
                            Row = 3,
                            Column = 22,
                            VioHandle = 0);

  ErrorCode = VIOWRTCHARSTR(CharStr = "(Press q to Quit)",        /* Display how to exit program  */
                            VioLength = strlen(CharStr),
                            Row = 23,
                            Column = 30,
                            VioHandle = 0);

  ErrorCode = VIOWRTCHARSTRATT(CharStr = "PROCESS 1",             /* Label area of screen used    */
                               VioLength = strlen(CharStr),       /* for Process 1                */
                               Row = 5,
                               Column = 16,
                               Attribute = "\034",
                               VioHandle = 0);

  for(RowCounter = 0; RowCounter < Window1Length; RowCounter++)   /* Display Thread 1 window      */

     ErrorCode = VIOWRTCHARSTRATT(Thread1Window[RowCounter],
                                  VioLength = strlen(Thread1Window[RowCounter]),
                                  Row = 7 + RowCounter,
                                  Column = 19,
                                  Attribute = "\034",
                                  VioHandle = 0);

  for(RowCounter = 0; RowCounter < Window2Length; RowCounter++)   /* Display Thread 2 window      */

     ErrorCode = VIOWRTCHARSTRATT(Thread2Window[RowCounter],
                                  VioLength = strlen(Thread1Window[RowCounter]),
                                  Row = 20 + RowCounter,
                                  Column = 19,
                                  Attribute = "\034",
                                  VioHandle = 0);

  ErrorCode = VIOSETCURPOS(Row = 21,                              /* Place cursor inside the      */
                           Column = 20,                           /* Thread 2 window              */
                           VioHandle = 0);
```

תרשים 18. תהליכים רבים: תהליך 1 (חלק 3) - אתחול ותצוגה

אם המחרוזת ארוכה יותר מאשר השורה הנוכחית, השארית של המחרוזת עוברת לשורה הבאה עד אשר מגיעים לסוף המסך. ברגע שהפעולה מגיעה לסוף המסך הפונקציה מסיימת.

VioSetCurPos

VioSetCurPos קובעת את המיקום של הסמן. לפונקציה זו יש שלושה פרמטרים.

```
ErrorCode = DOSMAKEPIPE((unsigned far *)&ReadHandle,             /* Create the pipe that will    */
                        (unsigned far *)&WriteHandle,            /* be inherited by Process 2    */
                        PipeSize = 16);

for(StringIndex = 0; StringIndex <= strlen(PgmPointer); StringIndex++)   /* Put the child process program*/
   ArgumentString[StringIndex] = ProgramName[StringIndex];       /* name in the first argument   */
                                                                 /* string                       */
ArglIndex = StringIndex;

HandleStringPointer = itoa(ReadHandle,HandleString,Radix);       /* Convert the read handle to a */
                                                                 /* string                       */
for (StringIndex = 0; StringIndex <= strlen(HandleStringPointer); StringIndex++)
   ArgumentString[ArglIndex + StringIndex] = HandleString[StringIndex];  /* Make read handle string the */
                                                                 /* second argument string       */

ErrorCode = DOSEXECPGM(ObjNameBufPointer,                        /* Exec Process 2               */
                       ObjNameLength,
                       ExecFlags,
                       ArgPointer,
                       EnvPointer,
                       ReturnCodesAddress,
                       PgmPointer);

ErrorCode = DOSSEMSET(SemHandle);                                /* Initialize semaphore as set. */

ErrorCode = DOSCREATETHREAD(KeyboardProcedure,                   /* Create thread 2 for keyboard */
                     (unsigned far *)&ThreadIDWord,              /* procedure.                   */
                     (unsigned char far *)&NewThreadStack[98]);

DisplayProcedure();                                              /* Call the display procedure   */
                                                                 /* with Thread 1                */
ErrorCode = VIOSETCURPOS(Row = 24,
                         Column = 0,                             /* Move the cursor before exit  */
                         VioHandle);

printf("Issuing DosExit with ActionCode set to terminate all threads.");  /* Display the ending status */
                                                                 /* message.                     */
DOSEXIT(ActionCode = 1,
        ResultCode = 0);                                         /* Notify OS/2 of termination   */

}                                                                /* End of C main routine.       */
/**********************************************************************************************/
```

תרשים 19. תהליכים רבים: תהליך 1 (חלק 4) - שאר הפונקציות בפרוצדורה Main

Row הוא ערך בגודל מלה, המכיל מספר מבוסס-אפס של השורה שבה יש להציב את הסמן.

Column הוא ערך בגודל מלה, המכיל מספר מבוסס-אפס של הטור שבו יש להציב את הסמן.

VioHandle שמור לשימוש עתידי וחייב להכיל מלה של אפסים.

DosMakePipe

הפונקציה יוצרת צינור לצורך תקשורת בין תהליכים. לפונקציה זו שלושה פרמטרים.

ReadHandle הוא מחוון רחוק לכתובת של מלה, שאליה מוחזר מקשר הקריאה לצינור. מקשר זה משמש לפונקציות אחרות כדי לקרוא נתונים מהצינור.

WriteHandle הוא מחוון רחוק לכתובת של מלה, שאליה מוחזר מקשר הכתיבה לצינור. מקשר זה משמש לפונקציות אחרות כדי לכתוב נתונים לצינור.

PipeSize הוא מלה המכילה את מספר הבתים שהמערכת שומרת לנתונים בצינור.

```
void DisplayProcedure()                                            /* Display procedure executed by*/
{                                                                  /* Thread 1                     */
  do{
    ErrorCode = DOSSEMREQUEST(SemHandle,                           /* Block on the SemRequest until*/
                              Timeout);                            /* a key press causes a         */
                                                                   /* SemClear in Thread 2         */
    NewFillChar[0] = FillChar[0];
    if (NewFillChar[1] > 0x70) NewFillChar[1] = 0x0F;              /* Calculate next display       */
    else NewFillChar[1] = NewFillChar[1] + 0x10;                   /* attribute to use             */

    ErrorCode = VIOSCROLLUP(TopRow=8,                              /* Display new character and    */
                            LeftCol=20,                            /* attribute                    */
                            BotRow=17,
                            RightCol=20,
                            NumLines=1,
                            (char far *)NewFillChar,
                            VioHandle = 0);

    ErrorCode = DOSWRITE(WriteHandle,                              /* Write the character and      */
                         (char far *)NewFillChar,                  /* attribute to the pipe so that*/
                         BufferLength,                             /* Process 2 gets it            */
                         (unsigned far *)&BytesWritten);
                                                                   /* Repeat until a "q" character */
  }while(KeyStructure.char_code != 0x71);                          /* is received                  */
  return;
}                                                                  /* End of Display procedure     */
/*********************************************************************************************/

void far KeyboardProcedure()                                       /* Keyboard procedure.          */
{
  int ToggleSwitch = 0;
  for(;;){                                                         /* Infinite loop blocks waiting */
    ErrorCode = KBDCHARIN(KeyStructurePointer,0,0);                /* for a character and then     */
                                                                   /* signals the Display procedure*/
    FillChar[0] = KeyStructure.char_code;                          /* via SemClear                 */
    if(ToggleSwitch == 0){
      FillChar[1] = 0x0F;                                          /* ToggleSwitch switches between*/
      ErrorCode = VIOSCROLLUP(21,20,21,20,1,(char far *)FillChar,0); /* normal and inverse video to  */
      ToggleSwitch = 1;                                            /* make it easier to see        */
    }                                                              /* Thread 2 reading keyboard    */
    else{                                                          /* input                        */
      FillChar[1] = 0x70;
      ErrorCode = VIOSCROLLUP(21,20,21,20,1,(char far *)FillChar,0); /* This thread terminates as a  */
      ToggleSwitch = 0;                                            /* result of Thread 1 issuing   */
    }                                                              /* a DosExit with ActionCode set*/
                                                                   /* to 1                         */
    ErrorCode = DOSSEMCLEAR(SemHandle);
  }
}                                                                  /* End of Keyboard procedure.   */
/*********************************************************************************************/
```

תרשים 20. תהליכים רבים: תהליך 1 (חלק 4) - פונקציות התצוגה והמקלדת

DosWrite

פונקציה לטיפול בקבצים, שתפקידה לכתוב נתונים לקובץ. השימוש בפונקציה זו תופס גם לגבי צינורות וגם לגבי התקנים התומכים במישק מערכת הקבצים (כמו, DosWrite ל-LPT1, כאשר LPT1 הוא שם שמור למדפסת מספר 1). הפונקציה מעבירה מספר מוגדר של בתים ממאגר הפלט לקובץ/צינור/התקן ואז חוזרת ל-thread המשתמש בה.
לפונקציה זו ארבעה פרמטרים.

FileHandle הוא ערך בגודל מלה המכיל את המקשר לקובץ או להתקן שהתקבל מה-DosOpen, או מקשר בכתיבה לצינור שהוחזר לאחר השימוש ב-DosMakePipe.

BufferArea הוא מחוון רחוק למאגר (buffer) הפלט של המשתמש בפונקציה.

BufferLength הוא מלה המכילה את מספר הבתים שיכתבו מהמאגר.

BytesWritten הוא מחוון רחוק לכתובת של מלה שבה ייכתב מספר הבתים שנכתבו באמת.

```
/*********************************************************************************************************/
/*  Multiple Processes with Multiple Threads - Process 2                                                 */
/*********************************************************************************************************/

#include <doscall.h>                                          /* OS/2 API dynamic link library*/
#include <stdio.h>                                            /* C standard I/O run time lib  */
#include <string.h>                                           /* C string library             */

/* Display window character definitions ******************************************************************/

  char far *Proc2Window[12] = {"┌──────────────────────┐",       /* An array of strings used to  */
                               "│                      │",       /* outline the Process 1 display*/
                               "├──────────────────────┤",       /* window                       */
                               "│                      │",
                               "│                      │",
                               "│                      │",
                               "│                      │",
                               "│                      │",
                               "│                      │",
                               "│                      │",
                               "│                      │",
                               "└──────────────────────┘"};

  int             RowCounter = 0;                             /* Variables used in "for" loop */
  int             WindowLength = 12;

/* General Variables - used throughout the program *******************************************************/

  unsigned        ErrorCode;                                  /* Error code return from OS/2  */
                                                              /* function calls               */
/* VioWrtCharStrAtt Variables - used to display text prompts and messages ********************************/

  char far        *CharStr;                                   /* String to be written         */
  unsigned        VioLength;                                  /* Length of character string   */
  unsigned        Row;                                        /* Starting position - row      */
  unsigned        Column;                                     /* Starting position - column   */
  char far        *Attribute;                                 /* Display attribute            */
  unsigned        VioHandle = 0;                              /* Reserved word of zeros       */

/* VioScrollUp Variables *********************************************************************************/

  unsigned        TopRow;                                     /* Upper left hand corner       */
  unsigned        LeftCol;
  unsigned        BotRow;                                     /* Bottom right hand corner     */
  unsigned        RightCol;
  unsigned        NumLines;                                   /* Number of lines to scroll    */
  char            FillChar[2] = {0x20,0x1F};                  /* Fill character to use        */

/* DosRead Variables *************************************************************************************/

  unsigned        ReadHandle;                                 /* File handle(pipe read handle)*/
  char            CharacterCell[2];                           /* Input buffer                 */
  unsigned        BufferLength = 2;                           /* Input buffer length          */
  unsigned        BytesRead;                                  /* Bytes read - returned        */

/* DosGetDateTime Variables ******************************************************************************/

  struct DateTime      CurDateTime;                           /* Date and Time structure      */
  struct DateTime far *CurDateTimePointer = &CurDateTime;     /* Date/Time structure pointer  */
```

תרשים 21. תהליכים רבים: תהליך 2 (חלק 1) - הכרזות

אם קובץ או התקן נפתחו עם מאפיין של קריאה בלבד, הפונקציה DosWrite לא תתבצע.

DosRead

פונקציה זו שייכת למערכת הקבצים המשמשת לקריאת נתונים מקובץ. השימוש בה תופס גם לגבי צינורות וגם לגבי התקנים התומכים במישק מערכת הקבצים (כמו, DosRead מ-COM1, כאשר COM1 הוא שם שמור לכניסה הטורית מס' 1). הפונקציה מעבירה מספר מוגדר של בתים מהקובץ/צינור/התקן ל-buffer, ואז חוזרת ל-thread המשתמש בה.
לפונקציה זו ארבעה פרמטרים.

```
/* DosSleep Variables ********************************************************************************/

  unsigned long        TimeInterval;                                 /* Sleep time duration        */

/* String variables ***********************************************************************************/

  char                 TimeHour[3];                                  /* String variables used to   */
  char                 *TimeHourPointer = TimeHour;                  /* build the time string      */
  char                 TimeMinutes[3];
  char                 *TimeMinutesPointer = TimeMinutes;
  char                 TimeSeconds[3];
  char                 *TimeSeconds`~inter = TimeSeconds;
  char                 Time[9] = "  :  :  \0";
  char                 *TimePointer = Time;

  char                 DateDay[3];                                   /* String variables used to   */
  char                 *DateDayPointer = DateDay;                    /* build the date string      */
  char                 DateMonth[3];
  char                 *DateMonthPointer = DateMonth;
  char                 DateYear[5];
  char                 *DateYearPointer = DateYear;
  char                 Date[9] = "  /  /  \0";
  char                 *DatePointer = Date;

  char                 LogString[20];                                /* This string is output by the */
  char                 *LogStringPointer = LogString;                /* thread that is logging the . */
                                                                     /* event                      */
  int                  Radix = 10;                                   /* Used by the itoa function  */

/* DosExit Variables **********************************************************************************/

  unsigned             ActionCode=1;                                 /* Exit all threads in process */
  unsigned             ResultCode=0;                                 /* Result saved for DosCWait  */

/*****************************************************************************************************/

  void far             DateTimeProcedure();                          /* Function declarations      */
  void                 BuildTimeString();
  void                 BuildDateString();

/*****************************************************************************************************/
```

תרשים 22. תהליכים רבים: תהליך 2 (חלק 2) - הכרזות

FileHandle הוא ערך בגודל מלה המכיל את המקשר לקובץ או להתקן שהתקבל מה-DosOpen, או מקשר הקריאה מהצינור שהוחזר לאחר השימוש ב-DosMakePipe.

BufferArea הוא מחוון רחוק למאגר הקלט של המשתמש בפונקציה זו.

BufferLength הוא מלה המכילה את מספר הבתים שיקראו לתוך המאגר.

BytesRead הוא מחוון רחוק לכתובת של מלה שבה יירשם מספר הבתים שנקראו באמת.

אין בטחון בכך שאמנם נקרא את מספר הבתים שביקשנו לקרוא. לאחר השימוש בפונקציה זו, התכנית צריכה תמיד להשוות בין BufferLength לבין BytesRead.

DosExecPgm

פונקציה זו מאפשרת לתכנית לבצע תכנית אחרת כתהליך בן, ושתיהן יכולות להתבצע בצורה מסונכרנת, או ללא תלות זו בזו. אם שני התהליכים עומדים לרוץ בצורה עצמאית, התהליך החדש יכול לרוץ באותו session, או בצורה נפרדת. לפונקציה זו שבעה פרמטרים.

```
main(argc, argv, envp)                                          /* Start of C main routine       */
int argc;                                                       /* argc is count of argument     */
char *argv[ ];                                                  /* strings                       */
char *envp[ ];                                                  /* argv array contains argument  */
                                                                /* strings                       */
{                                                               /* envp is environment strings   */
/* DosCreateThread Variables ***********************************************************************/

  unsigned           ThreadIDWord;                              /* New thread ID                 */
  unsigned char      NewThreadStack[1002];                      /* New thread stack              */

/* Start of executable program *********************************************************************/

  ErrorCode = VIOWRTCHARSTRATT(CharStr = "PROCESS 2",           /* Label area of screen used     */
                          VioLength = strlen(CharStr),          /* for Process 2                 */
                          Row = 5,
                          Column = 55,
                          Attribute = "\032",
                          VioHandle = 0);

  for(RowCounter =0; RowCounter < WindowLength; RowCounter++)   /* Display the Process 2 window */
    ErrorCode = VIOWRTCHARSTRATT(Proc2Window[RowCounter],
                          VioLength = strlen(Proc2Window[RowCounter]),
                          Row = 7 + RowCounter,
                          Column = 48,
                          Attribute = "\032",
                          VioHandle);

  ErrorCode = DOSCREATETHREAD(DateTimeProcedure,                /* Create Thread 2 for date and */
                          (unsigned far *)&ThreadIDWord,        /* time procedure               */
                          (unsigned char far *)&NewThreadStack[1000]);

                                                                /* Get the pipe read handle from*/
  ReadHandle = atoi(argv[1]);                                   /* the second argument string   */

  do{                                                           /* Top of loop that continues   */
                                                                /* reading character cells from */
    ErrorCode = DOSREAD(ReadHandle,                             /* the pipe until a "q"         */
                    (char far *)CharacterCell,                  /* character is received        */
                    BufferLength,
                    (unsigned far *)&BytesRead);

    ErrorCode = VIOSCROLLUP(TopRow=10,                          /* Scroll the window with the   */
                        LeftCol=49,                             /* new attribute                */
                        BotRow=17,
                        RightCol=69,
                        NumLines=1,
                        (char far *)FillChar,
                        VioHandle = 0);

    LogString[0] = CharacterCell[0];                            /* Build the log string         */
    LogString[1] = 0x00;
    LogStringPointer = strcat(LogStringPointer," ");
    LogStringPointer = strcat(LogStringPointer,TimePointer);
    LogStringPointer = strcat(LogStringPointer," ");
    LogStringPointer = strcat(LogStringPointer,DatePointer);

    ErrorCode = VIOWRTCHARSTRATT((char far *)LogStringPointer,  /* Display the new log string   */
                          VioLength = strlen(LogStringPointer),
                          Row = 17,
                          Column = 49,
                          Attribute = &CharacterCell[1],
                          VioHandle);
  }                                                             /* Test for a "q" character     */
  while(CharacterCell[0] != 0x71);
                                                                /* Notify OS/2 of termination   */
  DOSEXIT(ActionCode,
        ResultCode);
}
/**************************************************************************************************/
```

תרשים 23. תהליכים רבים: תהליך 2 (חלק 3) - הפונקציה Main

```
void far DateTimeProcedure()
{

  for(;;){                                                          /* Begin endless loop           */

     ErrorCode = DOSGETDATETIME(CurDateTimePointer);                /* Get the date and time        */

     BuildTimeString();                                             /* Call BuildTimeString         */

     ErrorCode = VIOWRTCHARSTRATT(TimePointer,                      /* Display the time string      */
                                  VioLength = strlen(TimePointer),
                                  Row = 8,
                                  Column = 51,
                                  Attribute = "\032",
                                  VioHandle);

     BuildDateString();                                             /* Call BuildDateString         */

     ErrorCode = VIOWRTCHARSTRATT(DatePointer,                      /* Display the date string      */
                                  VioLength = strlen(DatePointer),
                                  Row = 8,
                                  Column = 60,
                                  Attribute = "\032",
                                  VioHandle);

     ErrorCode = DOSSLEEP(TimeInterval = 1000);                     /* Sleep for about a second     */

  }                                                                 /* This thread terminates when  */
                                                                    /* the process terminates       */
}
/******************************************************************************************************/

void BuildTimeString()                                              /* This function formats the    */
                                                                    /* time suitable for display    */
{
  TimeHourPointer = itoa(CurDateTime.hour,TimeHour,Radix);          /* Format the hour              */
  if(strlen(TimeHourPointer) == 1){
     TimePointer[0] = 0x30;
     TimePointer[1] = TimeHourPointer[0];
  }
  else{
     TimePointer[0] = TimeHourPointer[0];
     TimePointer[1] = TimeHourPointer[1];
  }

  TimeMinutesPointer = itoa(CurDateTime.minutes,TimeMinutes,Radix); /* Format the minutes           */
  if(strlen(TimeMinutesPointer) == 1){
     TimePointer[3] = 0x30;
     TimePointer[4] = TimeMinutesPointer[0];
  }
  else{
     TimePointer[3] = TimeMinutesPointer[0];
     TimePointer[4] = TimeMinutesPointer[1];
  }

  TimeSecondsPointer = itoa(CurDateTime.seconds,TimeSeconds,Radix); /* Format the seconds           */
  if(strlen(TimeSecondsPointer) == 1){
     TimePointer[6] = 0x30;
     TimePointer[7] = TimeSecondsPointer[0];
  }
  else{
     TimePointer[6] = TimeSecondsPointer[0];
     TimePointer[7] = TimeSecondsPointer[1];
  }
}
/******************************************************************************************************/
```

תרשים 24. תהליכים רבים: תהליך 2 (חלק 4) -
DateTimeProcedure & BuildTimeString

ObjNameBuf הוא מחוון רחוק למאגר שאליו יוחזר מידע, אם הפונקציה **DosExecPgm** תיכשל. השם של תכנית היעד שתרמה לכשלון יוצב במאגר.

ObjNameBufL הוא מלה המכילה את האורך בבתים של **ObjNameBuf**.

```
void BuildDateString()                                        /* This function formats the    */
                                                              /* date suitable for display    */
{
  DateMonthPointer = itoa(CurDateTime.month,DateMonth,Radix);  /* Format the month            */
  if(strlen(DateMonthPointer) == 1){
     DatePointer[0] = 0x30;
     DatePointer[1] = DateMonthPointer[0];
  }
  else{
     DatePointer[0] = DateMonthPointer[0];
     DatePointer[1] = DateMonthPointer[1];
  }

  DateDayPointer = itoa(CurDateTime.day,DateDay,Radix);        /* Format the day              */
  if(strlen(DateDayPointer) == 1){
     DatePointer[3] = 0x30;
     DatePointer[4] = DateDayPointer[0];
  }
  else{
     DatePointer[3] = DateDayPointer[0];
     DatePointer[4] = DateDayPointer[1];
  }

  DateYearPointer = itoa(CurDateTime.year,DateYear,Radix);     /* Format.the year             */
  DatePointer[6] = DateYearPointer[2];
  DatePointer[7] = DateYearPointer[3];
}
/**************************************************************************************************/
```

תרשים 25. תהליכים רבים: תהליך 2 (חלק 5) - BuildDateString

ExecFlags אות-הסימון הזה היא ערך בגודל מלה המצביע על אחת מהדרכים שבהן התהליך ירוץ. האות מציין אחת מהאפשרויות הבאות:

(1) ריצה מסונכרנת, או לא-מסונכרנת, לתהליך האב;
(2) עם שמירה, או ללא שמירה, של קוד התוצאה של סיום התהליך;
(3) עבודה תחת מעקב (trace), בדומה למה שנעשה עם מנפה שגיאות (debbuger);
(4) ריצה ברקע (background) כתהליך נפרד;
(5) טעינה לזיכרון וציפיה להפעלה על-ידי ה-session manager.

ArgPointer הוא מחוון רחוק להתחלה של שתי מחרוזות ארגומנטים, אשר מועברים לתהליך החדש ומשמשות כשורת פקודה של OS/2. מתרגם שורת הפקודה "מבין" שהמחרוזת הראשונה מבין המחרוזות היא שם התכנית, והשניה מכילה תווים נוספים המוכנסים בשורת הפקודה.

EnvPointer הוא מחוון רחוק לקבוצה של מחרוזות המועברות לתכנית ומכילות מידע על הקונפיגורציה.

ReturnCodes הוא מחוון רחוק לכתובת בגודל של מלה כפולה המשמשת לאיחסון מידע המתקבל לאחר השימוש בפונקציה זו. בתהליכים אסינכרוניים, המלה הראשונה מכילה את המספר המזהה לתהליך הבן. בתהליכים מסונכרנים, המלה הראשונה מכילה את קוד סיום הפעולה של המערכת, והמלה השניה מכילה את ה-ResultCode המצוין על-ידי ה-thread האחרון בתהליך הבן שהפעיל את הפונקציה DosExit.

PgmPointer הוא מחוון רחוק למחרוזת תווים המכילה את שם הקובץ של התכנית העומדת להתבצע.

כאשר שני תהליכים מתבצעים בצורה מסונכרנת, הכוונה היא שה-thread המפעיל את DosExecPgm עובר למצב המתנה עד אשר תהליך הבן מסתיים. תהליך בן שרץ בצורה נפרדת, מכוון לרוץ ברקע ללא אפשרות שימוש במקלדת או

במסך. יוצא דופן הוא השימוש בפונקציה VioPopUp כדי לטפל במצבי שגיאה. תהליך בן הרץ בצורה אסינכרונית חייב לעשות זאת בתוך המגבלה של שני תהליכים המתחלקים באותו session. הדבר כולל שיתוף במשאבי המקלדת והמסך השייכים ל-session זה.

מערכת ההפעלה בונה טבלת מתארים מקומית (LDT) כדי לספק לתהליך החדש מרחב כתובות משלו. הטבלה מאפשרת לתהליך הבן להכנס למשאבים הניתנים בירושה, אשר נפתחו כבר על-ידי תהליך האב, כמו מקשרים לקובץ מסוים וצינורות. בדוגמה שלנו, תהליך הבן יורש את מקשר הקריאה לצינור.

DosSleep

פונקציה זו מפסיקה את פעולת היע"מ במהלך פרק הזמן שהוקצה לה, ומעבירה את ה-thread למצב המתנה לפרק זמן מוגדר.
לפונקציה זו פרמטר אחד.

TimeInterval הוא ערך בגודל של מלה כפולה המצביע על פרק הזמן באלפיות השניה שיעבור עד אשר ה-thread יחדש את פעולתו.

ערכי הזמן מעוגלים כלפי מעלה לכפולות של מחזורי הזמן בשעון. הפעולה תתבצע בדיוק של עד שני תקתוקים בשעון, בהתאם לביצוע של ה-threads האחרים במערכת.

תיאור של תכנית בעלת תהליכים רבים

על-פי הלוגיקה של התכנית הנראית בתרשים 15 נראה, שתהליך האב (תהליך 1) מאוד דומה לתהליך שבדוגמה הקודמת. הוספנו לדוגמה זו כמות משמעותית של פעולות ק/פ ושימוש בצבע בתצוגה, כדי לשפר את המימד הויזואלי של התכנית. התכנית מניחה שהתצוגה היא במצב התואם עם 80*25 color text mode של האביזר IBM CGA. על-פי הלוגיקה של התכנית, כל אחד מה-threads כותב לשטח אחר של המסך וכתוצאה מכך, שורות רבות בתכנית מוקדשות להכרזות וללוגיקה שיעזרו לארגן את המסך.

בדומה לדוגמה הקודמת, thread מס' 1 בתהליך 1 אחראי להתחלת כל פעולה אחרת ולאחר מכן הוא מבצע את פרוצדורת התצוגה. בפרוצדורה העיקרית של תהליך 1 אנו יוצרים את הצינור לפני שאנו מתחילים את תהליך 2, כדי שתהליך 2 יוכל ל"רשת" אותו. thread מס' 1 התחיל את thread מס' 2 כדי שיבצע את פרוצדורת המקלדת, ואח"כ קרא לפרוצדורת התצוגה. בדוגמה זו שיפרנו את פרוצדורת התצוגה. ל-thread מס' 1 המבצע את פרוצדורת התצוגה יש חלון, שדרכו הוא מעביר את התווים שהתקבלו כתוצאה מההקשות האחרונות. thread התצוגה מחשב מאפיין תצוגה חדש לכל תו חדש שמקישים. זוג זה של תו/מאפיין הוא חלק מההודעה על אירוע שאנו רוצים לרשום. כדי לרשום את הודעת האירוע, thread מס' 1 כותב את הזוג הנ"ל לצינור שעבר בירושה לתהליך 2. thread מס' 1 ממשיך להתבצע בלולאה שלו, ממתין לאתת ואז מציג את התווים החדשים שנכנסו עד אשר מתקבל התו "q". לאחר שהתקבל התו הוא חוזר מפרוצדורת התצוגה. הפעולה האחרונה המוטלת על thread מס' 1 היא לסיים כהלכה את תהליך 1, כך ש-thread מס' 2 יסיים גם הוא את פעולתו.

שאר החלקים של תהליך 1, פרוצדורת המקלדת, מתבצעים על-ידי thread מס' 2 בדרך דומה מאוד לדוגמה הקודמת שעסקה במספר threads בתהליך. ההבדל העיקרי בין שתי הדוגמאות הוא, ששני ה-threads כותבים נתונים למסך כדי לספק אינדיקציה ויזואלית של הביצוע הרב-משימתי שמתרחש. thread מס' 2, המבצע את פרוצדורת המקלדת, מחליף בין תצוגה רגילה (normal video) לבין תצוגה במהופך (reverse video) של התווים המוצגים, כדי לספק משוב על הפעילות שלו.

אם נעבור לחלק השמאלי של תרשים 15, נראה ש-thread מס' 1 של תהליך 2 נשאר בתוך הפרוצדורה הראשית. הוא מנקה את המסך ונכנס ללולאה הקוראת הודעות מהצינור, משרשרת את הודעה עם זמן ותאריך, ומבצעת הדמייה של רישום על-ידי כתיבת ההודעה למסך. פעולה זו נמשכת עד אשר התו "q" נקרא מתוך הצינור, והוא מצביע על כך שהתהליך צריך לסיים את פעולתו.

thread מס' 2 בתהליך 2 מתחזק את המחרוזות של הזמן והתאריך הנוכחיים. בנוסף לשימוש שנעשה במחרוזת זו לבניית הודעת הרישום, אנו מציגים את מחרוזת התאריך והזמן, כדי להצביע על הביצוע של thread זה.

בתרשימים 16 ו-17, כללנו את ההכרזות לתהליך 1. המשפט הראשון הניתן לביצוע נמצא בתרשים 18. הוא קורא לפונקציה VioScrollUp כדי לנקות את המסך. הפונקציות הבאות לאחר מכן, VioWrtCharStr ו-VioWrtCharStrAtt משמשות לכתיבת כותרות והכותרת הצבעונית "thread windows". החלונות של תהליך 1 הם בצבע אדום בהיר (034 באוקטלי) והחלונות של תהליך 2 (תהליך הרישום) הם בצבע ירוק בהיר. השתמשנו במערכים עם נתונים קבועים כדי לצייר את ריבועי החלונות ובלולאות מסוג "for" כדי להוציא את המערכים לתצוגה. הפונקציה VioSetCurPos משמשת למיקום הסמן בחלון המציג את הפעילות של ה-thread המבצע את פרוצדורת המקלדת.

לאחר שהמסך תוחל (initalized) לתהליך 1, התכנית מתכוננת ליצירת תהליך הבן הנראה בתרשים 19. השלב הראשון בהכנותינו הוא הקצאת צינור בעזרת DosMakePipe. תהליך הבן יורש את הגישה לצינור זה, ואנו מעבירים אליו את מקשר הקריאה לצינור בעזרת מחרוזת הארגומנטים. ודאי תזכור מהתיאורים הקודמים על מישקי התכנות, שתהליך האב מעביר מחרוזת ארגומנטים לתהליך הבן בעזרת פרמטר, שהוא מחוון רחוק השייך לפונקציה DosExecPgm. לשם הנוחות, המחרוזת הראשונה במחרוזת הארגומנטים היא שם התכנית, ולכן שם התכנית מועתק ראשונה לתוך מחרוזת הארגומנטים. השתמשנו בפונקציה **itoa** **(integer to ascii)** כדי להפוך את מקשר הקריאה למחרוזת תווים והוספנו אותה למחרוזת הארגומנטים. ברגע זה אנו מוכנים ליצור את תהליך 2 בעזרת DosExecPgm. הפרמטר ExecFlags ב-DosExecPgm מצביע על כך שאנו רוצים שתהליך 2 יתבצע בצורה לא מסונכרנת לתהליך 1, ולכן השליטה תחזור לתהליך 1 מיד לאחר שתהליך 2 מתחיל להתבצע. מה שנשאר לתאר מתהליך 1, מאוד דומה לדוגמה הקודמת: תהליך 1 יוצר את thread מס' 2 כדי שיבצע את פרוצדורת המקלדת, ולאחר מכן הוא קורא לפרוצדורת התצוגה בעזרת thread מס' 1.

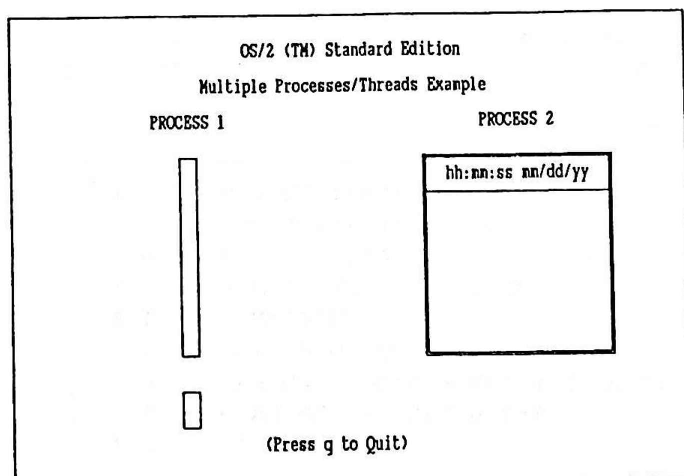
תרשים 20 מכיל את פרוצדורת התצוגה ואת פרוצדורת המקלדת. ניתן לראות שהלוגיקה הבסיסית שלהן נשארה ללא שינוי מהדוגמה הקודמת. הוספנו את הפונקציה DosWrite כדי לכתוב כל יחידת תו (כוללת את תו ה-ASCII ואת מאפיין התצוגה שלו) לצינור. לשתי הפרוצדורות הוספו קריאות לתת-מערכת התצוגה (vio), כך שניתן לראות את ביצוע שני ה-threads במסך.

תרשימים 21 ו-22 מכילים את ההכרזות לתהליך 2. בתרשים 23, תהליך 2 (תהליך הרישום שלנו) מאתחל את השטח שלו במסך על-ידי שימוש בטכניקות הדומות לאלה של תהליך 1. תהליך זה יוצר את thread מס' 2 המבצע את הפרוצדורה DateTime. לאחר שהוא שב מביצוע הפונקציה DosMakeThread, משיג thread מס' 1 את מקשר הקריאה לצינור ממחרוזת המשתנים שהועברה מתהליך 1. לאחר מכן נכנס thread זה ללולאה הקוראת יחידת תו מהצינור, מגולל כלפי מעלה את החלון שלו במסך, וכותב את מחרוזת הרישום החדשה. thread מס' 1 בודק אם התו "q" הגיע דרך הצינור. תו זה מצביע על כך שיש לסיים את פעולתו של התהליך עם כל ה-threads שלו.

תרשים 24 מראה את הפרוצדורה DateTime המבוצעת על-ידי thread מס' 2. פרוצדורה זו אחראית על תחזוקת המחרוזת של תאריך וזמן ועל הצגתה בחלק העליון של חלון התצוגה השייך לתהליך 2. פרוצדורת ה-DateTime קוראת לפונקציה DosGetDateTime, ואח"כ לשאר הפונקציות כדי לערוך את התאריך והזמן כמחרוזת תווים הניתנת להצגה במסך. thread זה משתמש גם בפונקציה DosSleep, כדי להשעות את הביצוע של עצמו ולעבור למצב של המתנה למשך שניה בערך. thread מס' 2 מפסיק לפעול כאשר thread מס' 1 מפעיל את DosExit עם ActionCode שווה ל-1.

הפונקציה BuildTimeString המוצגת בתרשים 24 ו-BuildDateString המוצגת בתרשים 25 עורכות את מחרוזת התאריך והזמן בהתאם למקובל בארה"ב. בפרק 7, העוסק בתכנות מתקדם נראה כיצד ניתן להכניס את שתי הפונקציות האלה לספריית הקישור הדינמי, כדי לאפשר לנו לתמוך בשימוש אחר, מבלי לשנות את תכנית היישום המשתמשת בפונקציות אלה.

תרשים 26 מתאר את התבנית של המסך כאשר מתבצעת תכנית הדוגמה בעלת תהליכים רבים.



תרשים 26. תהליכים רבים: תיאור מסך

תמיכה בזיכרון-יתר

לפני שנסיים פרק זה נתבונן בדוגמת תכנות נוספת, המתארת את היכולת של OS/2 לתמוך בזיכרון-יתר. היישום שלנו במקרה זה יכול להיות גיליון אלקטרוני גדול מאוד. אנו מאפשרים לך לציין את כמות הזיכרון הדרושה לגיליון אלקטרוני זה, ואף לציין יותר זיכרון מאשר עומד לרשותך במערכת. התכנית מבצעת הדמייה של גיליון אלקטרוני הנכנס לנתונים בכל אחד מהסגמנטים שלו המוקדש לנתונים.

כאשר אתה מפעיל את תכנית הדוגמה הזו משורת פקודה, אתה צריך גם להכריז על מספר הסגמנטים בני 64KB שאתה רוצה להקצות. ברגע שמוקצים הסגמנטים של הנתונים, אנו מתחילים להתייחס בזה אחר זה לכל סגמנט, ועל-ידי כך לבצע הדמייה של חישוב מחדש של הגיליון. אם גרמת לתכנית לתמוך בזיכרון-יתר, תראה שהנורה של הדיסק הקשיח מהבהבת. זה סימן שמנהל הזיכרון של OS/2 מעביר את הסגמנטים האלה לדיסק הקשיח ומחזיר אותם לזיכרון. הסגמנטים המועברים נקבעים בעזרת האלגוריתם המחשב את אלה שהיו הכי פחות בשימוש לאחרונה. דיון נרחב בנושא ראה בפרק 2 בסעיף העוסק באפשרויות ניהול הזיכרון.

כדי שדוגמה זו תפעל, אתה חייב לאפשר למערכת ההפעלה לבצע את העברות הסגמנטים. אם נתיב ההעברה הוא לדיסק הקשיח - זו תהיה ברירת המחדל, אלא אם עקפת אותה על-ידי המשפט MEMMAN=NOSWAP ב-CONFIG.SYS. אם קבעת שנתיב ההעברה הוא לדיסקט, מן הראוי שתדע שברירת המחדל היא **לא** לבצע את ההעברות. כדי לאפשר אותן, אתה חייב להוסיף את המשפט MEMMAN=SWAP ב-CONFIG.SYS. העברה לדיסקטים אינה מומלצת, בגלל איטיות הפעולה.

הפונקציות של OS/2

התכנית המדגימה תמיכה בזיכרון-יתר מוצגת בתרשימים 27 עד 29. בתכנית זו נשתמש בפונקציה נוספת של OS/2.

- השג כמות של זיכרון להקצאה (פרמטר בשורת הפקודה)
- הקצה את הסגמנטים
- קבע את מחוון הסגמנט הנוכחי
- בצע עד אינסוף
  - גע בסגמנט הנוכחי
  - הצג את מספר הסגמנט שכרגע נגעת בו
  - העלה את מחוון הסגמנט ב-1
- סיום לולאה
- סיים את התהליך

תרשים 27. הלוגיקה של תכנית הדוגמה - ניהול הזיכרון

```
/***************************************************************************************/
/*  Memory Overcommitment - Sample Program                                            */
/***************************************************************************************/

#include <doscall.h>                                /* OS/2 API dynamic link library*/
#include <stdio.h>                                  /* C standard I/O run time lib  */
#include <string.h>                                 /* C string library             */

/* General Variables ******************************************************************/

 unsigned          ErrorCode = 0;                   /* Error code return from OS/2  */
                                                    /* function calls               */

 unsigned far      *SelectorTable[256];             /* Array of far pointers        */

 int               SegmentCount = 0;                /* Loop variables               */
 int               TableIndex = 0;
 int               MaxIndex = 0;

 unsigned long     PointerBuilder;                  /* Used to build a far pointer  */

 unsigned          FakeDataValue;                   /* Used to touch a new segment  */

/* DosAllocSeg Variables **************************************************************/

 unsigned          Size = 0;                        /* Means size = 64K             */
 unsigned          Selector;                        /* Selector returned            */
 unsigned          AllocFlags = 0;                  /* Not sharable or discardable  */

/* DosExit Variables ******************************************************************/

 unsigned          ActionCode = 1;                  /* Exit all threads in process  */
 unsigned          ResultCode = 0;                  /* Result saved for DosCWait    */

/***************************************************************************************/
```

תרשים 28. תמיכה בזיכרון יתר (חלק 1) - הכרזות

DosAllocSeg

DosAllocSeg מקצה סגמנט בזיכרון עבור התהליך המשתמש בה. לפונקציה זו שלושה פרמטרים.

Size הוא ערך בגודל מילת מחשב המצביע על הגודל בבתים של הסגמנט שיוקצה. גודל הסגמנט יכול להיות בין 0 ל-65,536 בתים. כאשר הבקשה היא להקצות גודל 0, יוקצה סגמנט בן 64KB.

Selector הוא מחוון רחוק למלה שבה יוצב ערך הבורר. זהו ערך בר-תוקף לקריאה ל-LDT של התהליך, ולכן הוא מאפשר כניסה לזיכרון שהוקצה.

Flags הוא ערך בגודל מלה המראה אם מספר תהליכים יכולים להשתמש בסגמנט באמצעות הפונקציות DosGiveSeg ו-DosGetSeg, ואם ניתן לבטל אותו כאשר אין מספיק מקום בזיכרון.

אם לא הוגדר אחרת, הזיכרון שהוקצה בעזרת DosAllocSeg ניתן להזזה ולהעברה.

תכנית דוגמה לתמיכה בזיכרון-יתר

השלב הראשון בדוגמה דורש שנשיג את כמות הזיכרון שיש צורך להקצות. בשפת C, פרמטרים המוכנסים בשורת הפקודה מועברים לתכנית במשתנה-מערך-מחרוזת (string array variable) הידוע בשם "argv". החישוב של כמות הסגמנטים שיש להקצות נקבע בעזרת הפונקציה atoi (ASCII to integer) הפועלת על

האיבר הראשון בשורת הפקודה (argv[1]). לאחר מכן מציגה התכנית הודעת מצב הנותנת משוב על כך שמספר הסגמנטים שחשבנו להקצות, אמנם נכון.

לאחר מכן נכנסים ללולאה שמקצה סגמנטים בני 64KB. בשלב זה משתמשים בפונקציה DosAllocSeg תוך ציון הגודל כ-0, כדי להצביע על סגמנט בגודל 64KB. בסיום הפעולה התכנית מודיעה על כך במסך. בשלב הבא מכניסים את ערך הסגמנט המוחזר לתוך טבלה של מחוונים שמוקצית לכל סגמנט. יצירת מחוון רחוק בשפת C עם ערך הבורר המוחזר יכולה להיות משימה קצת מסובכת, ולכן אפשר להשתמש במספר טכניקות. בדוגמה זו השתמשנו בטכניקת ההיסט (shifting). ערך הבורר, שהוא מספר שלם בן 16 סיביות ללא סימן, הופך להיות מספר שלם ארוך בן 32 סיביות ללא סימן, הנקרא PointerBuilder. לאחר מכן מסיטים את ה-PointerBuilder 16 סיביות לשמאל, והופכים אותו למחוון רחוק בן 32 סיביות, המצביע על הבית הראשון בסגמנט. שימוש בטכניקה זו מפעיל את הודעת האזהרה "assignment to different types" של המהדר, אבל נשיג את התוצאה הרצויה.

לאחר שהקצינו את המספר הרצוי של הסגמנטים ובנינו מערך של מחוונים כדי לקרוא להם, נכנסנו ללולאה אינסופית הנכנסת לסגמנטים בזה אחר זה. אם אתה מקצה מספר סגמנטים הנכנסים אל שטח הזיכרון הפנוי כרגע, תראה שהתכנית מתבצעת מהר באופן יחסי. אך אם אתה מקצה מספר סגמנטים הגורם למצב של תמיכה בזיכרון-יתר, תראה שהדיסק פעיל מאוד והתכנית תרוץ לאט יותר, באופן משמעותי.

```
main(argc, argv, envp)                                        /* Start of C main routine    */
int argc;                                                     /* argc is count of argument  */
char *argv[ ];                                                /* strings                    */
char *envp[ ];                                                /* argv array contains argument */
                                                              /* strings                    */
{                                                             /* envp is environment strings */

  SegmentCount = atoi(argv[1]);                               /* Get # of segments to allocate*/
                                                              /* entered on command line and */
  printf("Number of 64K segments to allocate is %d\n",SegmentCount); /* output to the display */

  do {                                                        /* Loop to allocate segments  */

    ErrorCode = DOSALLOCSEG(Size,                             /* Allocate a 64K segment     */
                        (unsigned far *)&Selector,
                        AllocFlags);
    if (ErrorCode == 0)
       printf("Successfully allocated segment %d\n",TableIndex+1); /* Display status message */
    PointerBuilder = Selector;
    SelectorTable[TableIndex] = PointerBuilder << 16;         /* Put selector in pointer table*/
    ++TableIndex;
    --SegmentCount;
    }
  while (SegmentCount > 0);                                   /* End of allocation loop     */

  MaxIndex = TableIndex - 1;
  TableIndex = 0;

  for(;;) {                                                   /* Endless loop to touch      */
     FakeDataValue = *SelectorTable[TableIndex];              /* segments                   */
     printf("Just accessed segment %d \n",TableIndex+1);
     ++TableIndex;
     if (TableIndex > MaxIndex)
       TableIndex = 0;
  }

  DOSEXIT(ActionCode,                                         /* Notify OS/2 of termination */
          ResultCode);
}
/*****************************************************************************************************/
```

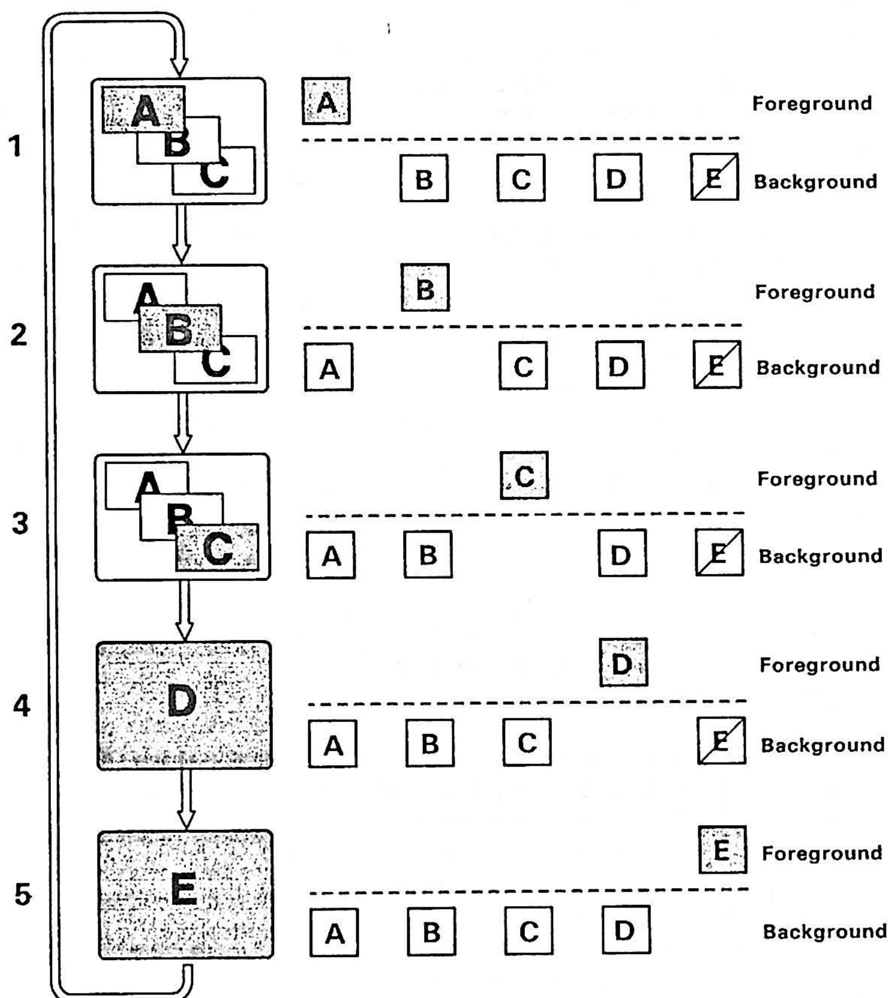
תרשים 29. תמיכה בזיכרון יתר (חלק 2) - פונקציה ראשית

לאחר שתבצע הידור וקישור לתכנית, תן לקובץ הניתן לביצוע את השם "allocmem.exe" כפי שאנו נתנו. ניתן להפעיל את התכנית משורת פקודה. אם תרצה להריץ את התכנית כך שתקצה 2MB של זיכרון, שורת הפקודה תיראה כך:

allocmem 32

בהתבסס על כמות הזיכרון במערכת, אתה צריך להריץ את התכנית פעמיים. בפעם הראשונה תרוץ כאשר יש מספיק זיכרון כדי לתמוך בבקשת הקצאת הזיכרון. בפעם השניה - הקצאת סגמנטים תכריח את המערכת לבצע תמיכה בזיכרון-יתר על-ידי ביצוע העברות.

E מציין יישום של DOS



יישומי PS/2 ויישומי DOS רצים בעת ובעונה אחת

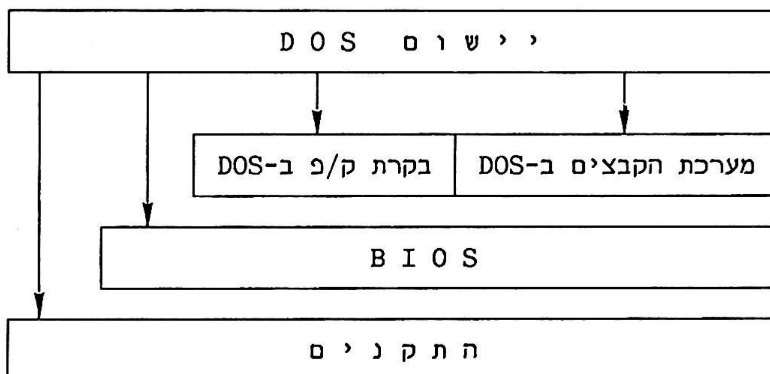
פרק 5

קלט ופלט על־ידי היישום

הטכנולוגיה של החומרה במחשבים מתקדמת בצעדי ענק וחשוב מאוד שנוכל לנצל אותה כראוי. השימוש במחשב ובהתקנים הצמודים אליו יכול להיות מוגבל בגלל היכולת של היישומים להשתמש בחומרה. כדי לאפשר ליישומים להשתמש בהתקנים השונים וכדי לספק פלטפורמה לצמיחה עתידית, OS/2 מספקת מגוון רחב של מישקים לתכנות יישומים (API) ומגוון של התקני קלט/פלט (ק/פ).

מה הופך את השיטות ש-OS/2 מציעה ליעילות כל כך עבור יישומים ותת-מערכות? היתרונות בולטים בהשוואה ל-DOS, שהיא סביבת תפעול חד-משימתית. יישום ב-DOS יכול להניח שכל התקני המערכת נמצאים בבעלותו והוא אינו מוגבל במה שהוא יכול לעשות. תרשים 30 מתאר את המישקים והמנגנונים לטיפול בהתקני ק/פ העומדים לרשות יישומי DOS.

יישום DOS יכול לשלוט בצורה ישירה על התקן באמצעות פקודות IN ו-OUT לכניסת הק/פ שלו (port). שימוש ישיר בק/פ הופך את היישום לתלוי במאפיינים של ההתקן. זאת אומרת, שיש סיכוי סביר שהוא לא יפעל אם ההתקן יוחלף. בנוסף לכך, ייתכן ותהיה בעיה ב-DOS כאשר השליטה של היישום בהתקן תופר על-ידי תכנית הנמצאת דרך קבע בזיכרון המערכת (-Terminate and-Stay-Resident - TSR). כדי להמנע מבעיות אלה, יישום יכול להשתמש במישקים אחרים במערכת: מערכת הקבצים של DOS, בקרת הק/פ (- I/O Control LOCtl) וה-BIOS. מצד אחד מישקים אלה מספקים את היתרון של אי-תלות בהתקן, אך מצד שני הם עדיין חשופים להפרעות מצד תכניות TSR. כאשר יש בעיה עם מישק שהופרע על-ידי תכנית TSR, קשה לקבוע איזו מן התכניות היא מקור התקלה.



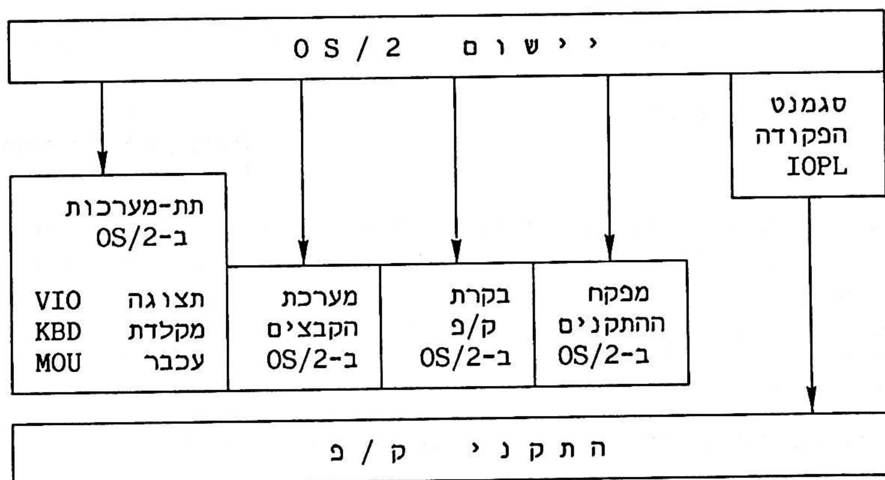
תרשים 30. מישקים להתקני ק/פ ב-DOS

לעומת זאת, אפשרויות הק/פ ב-OS/2 מספקות ליישום גמישות וסדר, שהם תוצאה ישירה של החופש לבחור. מפתח יישומים יכול לבחור מבין הדברים הבאים העומדים לרשותו ב-OS/2:

- מישקי מערכת הקבצים.
- מישקי בקרת הק/פ (IOCtl).
- מישקי תת-המערכות לתצוגה, למקלדת ולעכבר.
- מנגנון הפיקוח על התקני תווים.
- מנגנון סגמנט הפקודה עם IOPL.

מערכת הקבצים תומכת בק/פ למגוון של יעדים: קבצים, התקני תווים וצינורות. ה-IOCtl תומך בפונקציות המאפשרות בקרה על מספר רב של סוגי התקנים. מישקי התת-מערכת משפרים את יכולת היישום להתמודד עם התקני הקונסולה (consol: מסך, מקלדת, עכבר). מנגנון הפיקוח הינו טכניקה כללית לטיפול בנתוני ההתקנים. מנגנון סגמנט הפקודה של IOPL, הוא שיטה לביצוע ק/פ ישיר לכניסות של ההתקנים. שילוב של אמצעים אלה הינו שיפור משמעותי לעומת DOS. תרשים 31 מראה את השיטות העומדות לרשות יישום ב-OS/2 לביצוע ק/פ.

סדר ביצוע הק/פ הינו תוצאה ישירה של הסביבה רבת המשימות, שבה משאבי המערכת מתחלקים בין מספר יישומים. בסביבה כזאת יש צורך בכללים לק/פ, כך שהמשתמשים בהתקנים בו-זמנית, יוכלו "לחיות בשלום" זה עם זה. המישקים והמנגנונים של OS/2 כופים את כללי הגישה האלה היכן שאפשר. במספר מקרים, ליישום עצמו יש אמצעי בקרה המאפשר שליטה על הדרך, שבה יישום אחר יוכל לפנות להתקן שנמצא כבר בשימוש. לדוגמה, מערכת הקבצים מאפשרת ליישום לציין האם ואיך אפשר להתחלק בקובץ עם משתמשים אחרים. כאשר יישום יכול לגשת ישירות לכניסה של התקן כלשהו, הוא חייב למלא אחר כללים מוסכמים שיאפשרו לו לטפל בהתקן, אפילו אם הוא משתתף בו עם יישומים אחרים. להתקן כמו מסך, אשר ק/פ ישיר נעשה בו לעתים קרובות, יש תת-מערכת תצוגה עם פרוטוקול ק/פ ומספר מישקים. מישקים אלה מאפשרים ליישום לטפל בצורה ישירה במתאם התצוגה, ואינם מונעים את השימוש במסך על-ידי מספר יישומים באותו זמן.



תרשים 31. מישקי התקשרות עם התקני ק/פ

התקנים נפוצים	מישקים/מנגנונים לק/פ				
	מערכת קבצים	תת-מערכת	בקרת ק/פ	מנגנון פיקוח	IOPL
דיסק	כן	מערכת קבצים	כן	לא	לא
דיסקט	כן	מערכת קבצים	כן	לא	לא
מסך	כן	VIO	לא	לא	כן
מקלדת	כן	KBD	כן	כן	לא
עכבר	כן	MOU	כן	כן	לא
מדפסת	כן	Spooler	כן	כן	לא
תקשורת א-סינכר.	כן	לא	כן	לא	לא
רמקול	לא	(DosBeep)	לא	לא	כן
שעון	כן	שירותי שעון	לא	לא	לא

תרשים 32. ביצוע פעולות ק/פ ב-OS/2

אפשרויות הק/פ ב-OS/2 נותנות רמה בסיסית של תפקוד למספר התקנים, בסיוע של ה-device drivers המסופקים יחד עם מערכת ההפעלה. תרשים 32 מתאר את התועלת במישקי הק/פ ובמנגנונים השונים. הטבלה מתייחסת להתקני ק/פ נפוצים הנתמכים על-ידי OS/2 וה-device drivers שלה. מפתחי יישומים ותת-מערכות יכולים להתאים לרצונם פונקציות עבור יחידות ק/פ מסוימות, באמצעות הכלים של המערכת.

מערכת ניהול הקבצים

מבחינה תפקודית, מערכת התוכנה לניהול הקבצים ב-OS/2 הינה מרכיב חשוב בביצועי הק/פ. היא מאפשרת ליישום לבצע את הדברים הבאים:

- לפנות לסוגים רבים של התקנים.
- לשתף גישה להתקנים, או להגביל את הכניסה אליהם.
- לנתב מחדש ק/פ מהתקן אחד למשנהו.

בנוסף לכך, יש מספר יתרונות הנמצאים מעבר לאפשרויות הביצוע של מישקי מערכת ניהול הקבצים:

- תאימות עם DOS.
גישה ושימוש בקבצים עם שירותי מערכת ניהול הקבצים של OS/2 דומה לטיפול בקבצים עם DOS, מכיון ששתי מערכות הקבצים תואמות ביניהן. הכוונה בתאימות היא, שאתה יכול ליצור קובץ ב-DOS ולהשתמש בו ב-OS/2. וגם להיפך, תוכל ליצור קובץ ב-OS/2 ולהשתמש בו ב-DOS. תכונה זו חשובה ביותר לתאימות המערכות ועבור הגירה של יישומי DOS ל-OS/2.

- מנגנון שיתוף הקבצים נמצא באופן קבוע בזיכרון (resident).
אחד ההבדלים העיקריים בין מערכת ניהול הקבצים של OS/2 ובין זו של DOS הוא הזמינות של מנגנון שיתוף הקבצים. ב-OS/2 שיתוף קבצים מבוצע בצורה אוטומטית על-ידי התוכנה הנמצאת דרך קבע בזיכרון. ב-DOS אתה חייב להריץ קודם את התכנית SHARE.EXE לפני שמערכת ניהול הקבצים תוכל לבצע את שיתוף הקבצים. זאת אומרת, שלמערכת הקבצים ב-OS/2 יש **תמיד** את היכולת לבקר איך יישומים יפנו לקבצים. לגבי מערכת הקבצים ב-DOS, שיתוף הקבצים יבוא, או שלא יבוא, לידי ביטוי כתלות בהרצה של תכנית השירות SHARE.EXE על-ידי משתמש הקצה.

- ניהול אמצעי אחסון נתיקים.
הבדל נוסף בין מערכת הקבצים של OS/2 לבין זו של DOS הוא היכולת לנהל אמצעי אחסון נתיקים. מערכת הקבצים של OS/2 עורכת מעקב אחר אמצעי האחסון הנתיקים במטרה להבטיח את שלמות קבצי הנתונים. לדוגמה, אם משתמש הקצה הוציא את הדיסקט לפני שהיישום סיים לעבוד, OS/2 תורה למשתמש להחזיר את הדיסקט המסוים לכונן. בסביבת תפעול רבת משימות, יש סיכוי רב שיתרחשו פעולות ק/פ רבות. מנגנון ניהול יחידות האחסנה ההיקפיות (כרכים - volumes) עוקב אחר השימוש בהן ויודע איזה מבין הכרכים הנתיקים נמצא בשימוש.

בדרך כלל, מקובל לחשוב שמערכת ניהול הקבצים הינה מנגנון לביצוע פעולות קלט/פלט בהתקנים לאחסון קבצים. אולם, ניתן להשתמש במערכת זו לביצוע פעולות ק/פ גם בהתקני תווים כמו המדפסת, או קווי תקשורת אסינכרוניים. התקנים אלה נקראים התקנים סטנדרטיים (Standard Devices). מערכת הקבצים משתמשת בק/פ גם לגבי הצינור, שהוא קובץ מדומה המשמש בתקשורת בין תהליכים. רשימת ההתקנים שמערכת הקבצים יכולה לטפל בהם כוללת אפילו סוג מיוחד של התקן המאחסן קבצים - כונן דיסק/דיסקט לוגי. בהמשך, נדון בשימושים השונים של מערכת הקבצים עבור הסוגים השונים של ההתקנים.

התקנים המאחסנים קבצים

התקנים המאחסנים קבצים הם בד"כ דיסק קשיח או דיסקט. מערכת הקבצים היא למעשה תת-מערכת המנהלת התקנים המאחסנים קבצים. מערכת הקבצים שולטת על סוג הנתונים שיאוחסן ועל המיקום שלו בהתקן ומכתיבה את הצורה שבה יוכשר ההתקן כדי שיוכל לאחסן נתונים. זאת למעשה פעולת הפירמוט (format). מערכות הקבצים ב-OS/2 וב-DOS משתמשות בטבלה להקצאת קבצים (File Allocation Table - FAT) לשם ניהול הנתונים בהתקן. לגבי הדיסקט זאת אומרת שאתה מריץ את תכנית השירות FORMAT. דיסק קשיח יש לחלק תחילה למחיצות (עם תכנית השירות FDISK), ולאחר מכן לפרמט את המחיצה הרצויה. מערכת הקבצים מקצה אות אחת או יותר לכונן של ההתקן. אות זו משמשת

להגדרת התקן לוגי. אם במערכת יש שני כונני דיסקטים, היא תקצה לכל כונן אות זיהוי אחת נפרדת. מערכת הקבצים יכולה לייעד לכונן הדיסקטים עד שתי אותיות כונן. במקרה שיש רק כונן אחד, מערכת הקבצים תייעד לו את האותיות "a:" ו-"b:", שייצגו שני כוננים לוגיים נפרדים. הדבר מאפשר להעביר נתונים בין שני דיסקטים, למרות שיש רק כונן פיסי אחד. התקן המאחסן קבצים, כמו דיסק קשיח, ניתן לחלוקה למספר מחיצות. כל מחיצה יכולה לתפקד כהתקן נפרד לאחסון קבצים הנקרא דיסק לוגי.

יישום צריך לטפל בהתקן לאחסון קבצים באמצעות אות הכונן בלבד. יישום פונה בעיקר אל קבצים ולא אל התקני האחסון. קובץ כולל קבוצה של נתונים המאוחסנים בהתקן לאחסון קבצים. כך, יישום משתמש באות הכונן כדי לבחור בהתקן לוגי, ובשם של קובץ כדי לזהות נתונים מסוימים. מערכת הקבצים משתמשת במידע זה כדי לאתר את הנתונים בהתקן לאחסון הקבצים.

התאימות של מערכות הקבצים ב-DOS וב-OS/2 מאפשרת עקביות בכללי מתן השמות לקבצים. שם קובץ ב-OS/2 הוא מחרוזת תווים שאחריה רשומים (לא חובה) נקודה וסיומת. שם קובץ ב-OS/2 הוא בעל המאפיינים הבאים:

- מחרוזת השם הראשי יכולה להיות באורך של 1 עד 8 בתים (תווים).
- אורך הסיומת, שאינה חובה, נע בין 1 ל-3 בתים (תווים).
- כל התווים הם תווי ASCII, פרט לתווים שאינם מותרים:-
  - רווחים.
  - תווי ASCII בערכים הקסדצימליים מתחת ל-20.
  - תווים מיוחדים: < > + = : | ; , . " / \ [ ]

בסביבת OS/2, כאשר שם קובץ ארוך יותר מהתבנית [8.3] הוא אינו מקוצץ, אלא גורם להודעת שגיאה. בסביבת ה-DOS, השם מקוצץ לתבנית [8.3] כדי שיהיה לפי הכללים. ב-DOS התווים '*' ו-'?' משמשים כתווים גלובליים. התו '?' מייצג תו בודד והתו '*' מייצג קבוצה של תווים שאינם מזוהים במפורש.

פרוטוקול הק/פ ב-OS/2 הוא "ק/פ מבוסס מקשרים", אשר זהה לפרוטוקול ב-DOS. מקשר הוא ערך בן 16 סיביות שהמערכת משתמשת בו כמזהה למידע על קובץ מסוים. יישום חייב לעשות את הדברים הבאים בפעולות קלט/פלט:

- לכונן קשר, או לפתוח (OPEN), קובץ קיים, או קובץ חדש, כדי לקבל מקשר לקובץ.
- לקרוא נתונים מהקובץ, או לכתוב אליו בעזרת המקשר לקובץ.
- לסגור את הקובץ (CLOSE), ועל-ידי כך לשחרר את המקשר לקובץ.

כדי לפתוח או ליצור קובץ עם מערכת הקבצים של OS/2, צריך לתת פרמטרים הזהים לאלה הדרושים במערכת הקבצים של DOS. בתרשים 33 תמצא סיכום של הפרמטרים, אשר מתייחסים לפעולות של פתיחה ויצירה של קובץ. תהליך של OS/2 מספק מחוון למחרוזת תווי ASCII המציינת את הכונן (לא חובה) ואת מסלול החיפוש במדריך (לא חובה), את שם הקובץ ובית עם ערך 00 בהקסה. כאשר כל החלקים של מחרוזת זו נמצאים, היא נקראת שם קובץ "תקין בהחלט" (fully qualified). כאשר קובץ נפתח, התהליך חייב לציין את אופן הגישה אליו, ובכך להודיע למערכת הקבצים על סוג הפעילויות שהתהליך מתעתד לבצע בקובץ (קריאה בלבד, כתיבה בלבד, קריאה/כתיבה). התהליך מציין עוד את אופן השיתוף. בכך הוא מודיע על סוג הפעילות שהתהליך מאפשר לתהליכים אחרים לבצע בו-זמנית בקובץ (גישה בו-זמנית אסורה, גישה בו-זמנית מותרת

לצורך קריאה, גישה בו-זמנית מותרת לצורך כתיבה, גישה בו-זמנית מותרת לכל מטרה). בתרשים 34 תמצא את ההשלכות השונות של ההרשאות לשיתוף קבצים. בזמן יצירת קובץ חייבים לציין מאפיין (attribute) לקובץ, אשר מצביע שהקובץ מיועד לקריאה בלבד, אם זהו קובץ סמוי, קובץ מערכת, קובץ ארכיב או שהשם מגדיר תת-מדריך.

מרגע שקובץ נפתח או נוצר, המקשר שהתקבל הוא המפתח לק/פ. כפי שתזכור, תהליך הינו יישות שבבעלותה נמצאים משאבי המערכת, וזה כולל את המקשרים לקבצים. זאת אומרת, שכל ה-threads בתהליך יכולים להשתמש באותו מקשר כדי לבצע ק/פ לקובץ. מאחר ולכל ה-threads בתהליך יש את היכולת להשתמש באותו מקשר לקובץ, הם חייבים להיות מתואמים בינם לבין עצמם כדי למנוע התנגשויות בפעולות ק/פ.

ק/פ לקובץ מבוצע בעזרת המקשר לקובץ. המערכת מתחזקת מחוון קריאה/כתיבה או מחוון ק/פ לקובץ, וקובעת אותו על הבית הראשון בזמן הפתיחה של הקובץ. יישום יכול לבצע ק/פ בקובץ בצורה סידרתית באופן אוטומטי, כי מערכת הקבצים מזיזה את מחוון הק/פ למקום בקובץ שבו הסתיימה בקשת הק/פ האחרונה. יישום יכול לבצע ק/פ בצורה אקראית על-ידי שינוי המקום שבו נמצא מחוון הק/פ הנוכחי (פעולה זו ידועה גם בשם "חיפוש לוגי"). התהליך יכול לבצע ק/פ בצורה אסינכרונית בעזרת שתי טכניקות: שימוש בפונקציות של מערכת הקבצים המטפלות בקריאה וכתיבה בצורה אסינכרונית, או שימוש ב-thread שיבצע ק/פ בזמן ש-threads אחרים בתהליך יבצעו פעילויות אחרות.

ריבוי המשימות והשלבים הקשורים בביצוע של בקשת ק/פ, אינם מתירים למערכת הקבצים להבטיח את ביצוע בקשות הק/פ לפי הסדר שהתהליך ביקש. לדוגמה, אם המאגר של היישום נמצא בסגמנט שהועבר החוצה, מערכת הקבצים תנסה לבצע בקשות ק/פ אחרות של threads אחרים בתהליך, אשר ניתן לבצען מיד. בנוסף לכך, ה-device driver של הדיסק עושה אופטימיזציה של ביצועי ההתקן על-ידי מיון הבקשות שהוא מקבל לפי מספר הסקטור הפותח, במטרה להפחית את הזמן שדרוש לאתר את הנתונים. אם תהליך תלוי בצורה קריטית במידע שעודכן בקובץ בסדר מסוים, אסור לו להתחיל ביצוע של מספר בקשות ק/פ בו-זמנית, אלא עליו להקצות thread אחד לביצוע כל בקשות הק/פ.

כאשר לתהליך אין עוד צורך לבצע ק/פ לקובץ, הוא סוגר את המקשר לקובץ. פעולה זו מסיימת את הקשר בין התהליך לבין הקובץ וגורמת למערכת הקבצים לעדכן את המידע במדריך ולנקות כל מאגר המכיל מידע הקשור לקובץ. במקרה שהיישום מסתיים בצורה לא שיגרתית, מערכת ההפעלה סוגרת כל מקשר לקובץ שהיה בבעלות התהליך (שים לב: המקשר הוא משאב של המערכת).

תהליך יכול לציין בזמן שהוא פותח או יוצר קובץ, אם תהליכי הבן שהוא מתחיל יכולים לרשת את המקשר לקובץ. כך, כאשר תהליך האב מתחיל את תהליך הבן, תהליך הבן יקבל העתק לכל מקשר קובץ הניתן לירושה והנמצא בבעלות התהליך. זאת אומרת, שק/פ המבוצע על-ידי תהליך הבן עם מקשר ק/פ שניתן בירושה, ישפיע על תהליך האב (ולהיפך), מכיון שמחוון הק/פ בקובץ יעודכן כתוצאה מק/פ המגיע משני מקשרים קיימים לקבצים. אולם, ניתן לסגור כל אחד מהמקשרים האלה בצורה בלתי תלויה בחברו. תהליך הבן יכול לסגור את המקשר לקובץ שלו, ותהליך האב יוכל להמשיך להשתמש במקשר שלו ולהיפך.

פרמטר	תיאור				
שם	שם הקובץ				
מצב (open FLAG)	קיים			לא קיים	
	פתח	החלף	כשלון	הגדר	כשלון
אופן הגישה (open MODE)	ירושה כתיבה ישירה שגיאות כשלון R, W, W/R DR, DW, DRW, DN			ירושה כתיבה ישירה שגיאות כשלון R, W, W/R DR, DW, DRW, DN	
גודל הקובץ		גודל		גודל	
מאפיין הקובץ (attribute)		קרא בלבד סמוי מערכת ארכיב מדריך		קרא בלבד סמוי מערכת ארכיב מדריך	

<u>אופן השיתוף</u>

DR = Deny Readers (כתיבה בלבד)
DW = Deny Writers (קריאה בלבד)
DRW = Deny All (אין כלל הרשאות)
DN = Deny None (קריאה וכתיבה)

<u>אופן הגישה</u>

R = קריאה
W = כתיבה
W/R = קריאה/כתיבה
write-through = כתיבה ישירה

תרשים 33. הפרמטרים של DosOpen עבור קבצים

סוג ההרשאה שניתנה	אופן הגישה המיועד		
	קריאה	כתיבה	קריאה/כתיבה
כתיבה בלבד	קורא אחד כותבים רבים	אין קוראים כותבים רבים	קורא אחד כותבים רבים
קריאה בלבד	קוראים רבים אין כותבים	קוראים רבים כותב אחד	קוראים רבים כותב אחד
לא קריאה ולא כתיבה	קורא יחיד אין כותבים	אין קוראים כותב יחיד	קורא יחיד כותב יחיד
הכל מותר	קוראים רבים כותבים רבים	קוראים רבים. כותבים רבים	קוראים רבים כותבים רבים

תרשים 34. ההשפעה של הרשאות על שיתוף קבצים

ירושה של מקשרי קבצים הינה הגורם העיקרי המאפשר הכוונה מחדש (redirecting) של ק/פ. תהליך האב יכול לקבוע למקשר עם מי לפעול, עם קובץ, או עם יעד אחר כלשהו. תהליך האב יכול לפתוח מקשר חדש, או לאלץ את המקשר הקיים להתאים את עצמו ליעד מסוים. לאחר מכן הוא יכול להתחיל תהליך בן שישתמש במקשר זה שעבר בירושה.

מגבלת המערכת היא שמספר מקשרי הקבצים לא יעלה על 255. לכל תהליך מוקצים בתחילה 20 מקשרים והוא יכול להגדיל את המספר הזה בעזרת פונקציה של המערכת. ההגדלה תלויה במספר התהליכים האחרים במערכת, ובמספר המקשרים שהוקצו להם.

התקני תווים

בדומה למערכת הקבצים של DOS, אפשר להשתמש במערכת הקבצים של OS/2 לביצוע ק/פ להתקני תווים. יש לציין שהתקן המכיל תו אינו דומה להתקן המכיל קובץ. מרגע שנשלחו נתונים להתקן תו או התקבלו ממנו, הם אינם ניתנים לשינוי. משלוח של נתונים להתקן תו אינה משנה את הנתונים שכבר נשלחו; קריאה מחדש של נתונים מהתקן אינה מביאה נתונים הזהים לקריאה הקודמת. זאת אומרת, שיישום חייב לטפל בנתונים בסדר מסוים. להתקני תווים יש גם מאפיינים מיוחדים, חלק מהם - התקני קלט (כמו המקלדת) שניתן לקרוא מהם נתונים בלבד וחלק מהם - התקני פלט בלבד (כמו מדפסת) שניתן לכתוב אליהם נתונים. ויש גם התקני ק/פ (כמו התקן לתקשורת טורית), כך שניתן לקרוא מהם ולכתוב אליהם.

השלבים הדרושים כדי לגשת להתקני תו ולהשתמש בהם עם מערכת הקבצים של OS/2 דומים לאלה הדרושים במערכת הקבצים של DOS. הפרוטוקול שבו מערכת הקבצים משתמשת לצורך קשר עם התקני התו מבוסס על מקשרי ק/פ. מקשר ההתקן דומה למקשר הקובץ. זהו ערך בן 16 סיביות שמערכת הקבצים משתמשת בו כמזהה ייחודי למידע על ההתקן. יישום חייב לעשות את הדברים הבאים:

1. לפתוח את ההתקן בעזרת שם ההתקן כדי לקבל מקשר להתקן.
2. לכתוב להתקן, או לקרוא ממנו בעזרת המקשר להתקן.
3. לסגור את ההתקן בעזרת המקשר להתקן.

כפי שצפוי, כללי מתן השמות להתקני התווים זהים ב-OS/2 וב-DOS. בשתי המערכות ה-device driver של התקן התו מספק את השם. שם התקן התו הינו מחרוזת של תווי ASCII הממלאת אחר הכללים הבאים הדומים לאלה של הקבצים:

- גודל המחרוזת הוא בין 1 ל-8 בתים.
- כל התווים הם תווי ASCII פרט לתווים שאינם מותרים:
  - רווחים.
  - תווי ASCII בערכים מתחת לערך ההקסדצימלי 20.
  - תווים מיוחדים: < > + = : | ; , . " / \ [ ]

OS/2 כוללת device drivers להתקני תו מסוימים, לשימוש של היישומים המופעלים בה. להלן ההתקנים ושמותיהם:

CON	התקני הקונסולה (מסך ומקלדת).
KBD$	מקלדת.
LPT1, LPT2, LPT3	שלוש מדפסות מקביליות.
NUL	התקן דמי.
PRN	שם נוסף למדפסת המקבילית הראשונה.
SCREEN$	מסך.

ניתן לפנות להתקני הקונסולה, למסך ולמקלדת באמצעות מערכת ניהול הקבצים של OS/2, אך השימוש בהם הוא בד"כ באמצעות תת-המערכות להתקני התצוגה והמקלדת.

כדי שתוכל להשתמש בהתקני תווים אחרים עם מערכת הקבצים, יש צורך להתקין device drivers בקובץ CONFIG.SYS בעזרת הפקודה =DEVICE.

כדי לפתוח התקן תו, היישום צריך לציין את שם ההתקן במקום שם הקובץ, ללא שם כונן וללא מסלול חיפוש. יישום אינו יכול להגדיר התקן תו, כי הדבר נעשה באמצעות ההתקנה של ה-device driver. בפקודת המערכת OPEN (גם ב-DOS וגם ב-OS/2), יש לשם ההתקן עדיפות על שם הקובץ. זאת אומרת, ששם הקובץ צריך להיות שונה משם ההתקן. בנוסף לכך, התהליך מציין את אופן הגישה, המגדירה את סוג הפעילות שהתהליך מתכוון לבצע בהתקן התו (קריאה בלבד, כתיבה בלבד, כתיבה/קריאה). התהליך צריך גם לציין את אופן השיתוף או סוג הפעילות, שהתהליך יאפשר לתהליכים אחרים לבצע בו-זמנית בהתקן התו (גישה בו-זמנית אינה מותרת, מותרת גישה בו-זמנית לקריאה בלבד, מותרת גישה בו-זמנית לכתיבה בלבד, או גישה בו-זמנית מותרת לכל צורך). בתרשים 34 תמצא את ההשלכות של ההרשאות השונות על שיתוף הקבצים.

ההרחבה של פונקציית שיתוף הקבצים במערכת הקבצים של OS/2 אשר כוללת התקני תו, הינה שיפור לעומת מערכת הקבצים של DOS. יתר על כן, בעזרת ה-device driver של התקן התו ניתן לציין זאת בקובץ ה-CONFIG.SYS.

פונקציית המערכת OPEN מחזירה מקשר המזהה את ההתקן למערכת הקבצים. בתרשים 35 תמצא סיכום של הפרמטרים הדרושים לפונקציה זו כדי שתוכל לפתוח התקן תו. בדומה למקשר הקובץ, גם מקשר ההתקן נמצא בבעלות התהליך. לכן, כל ה-threads בתהליך יכולים להשתמש במקשר כדי לבצע ק/פ להתקן והם חייבים להיות מתואמים בינם לבין עצמם כדי למנוע התנגשויות בביצוע פעולות ק/פ. בדומה למקשרי הקבצים, תהליך יכול לציין בזמן שהוא פותח את התקן התו, אם תהליכי הבן שלו יוכלו לרשת את המקשר להתקן. במלים אחרות, תהליך הבן יכול לקבל עותק של מקשר ההתקן המתאים לכל אחד ממקשרי ההתקנים הניתנים לירושה והנמצאים בבעלות התהליך. תהליך הבן יכול לבצע ק/פ להתקן התו מבלי לפתוח אותו. תהליכי הבן והאב חייבים לתאם את הק/פ להתקן, מכיון שאופי הנתונים הקשורים בהתקן התו הוא "קרא פעם אחת ו/או כתוב פעם אחת". אולם, ניתן לסגור כל אחד מהמקשרים ללא תלות במה שקורה לחברו. תהליך הבן יכול לסגור את מקשר ההתקן שלו, ותהליך האב יוכל להמשיך להשתמש במקשר שלו ולהיפך.

ניתן להשתמש במקשר ההתקן בפונקציות לקריאה וכתיבה כדי שהן יוכלו לבצע את הק/פ להתקן. יישום יכול לבקש שמספר בתים מסוים יועבר להתקן, ומערכת הקבצים ב-OS/2 תודיע ל-device driver של התקן התו שיבצע ק/פ עם המספר הזה של בתים. הנוהל הזה שונה ממערכת הקבצים של DOS, שבה אין הבדל בין בקשה המכילה בתים רבים לבין בקשה המכילה בית אחד בלבד. בדומה לקבצים, התהליך יכול לבצע ק/פ בצורה אסינכרונית בעזרת שתי טכניקות. הוא יכול

להשתמש בפונקציות של מערכת הקבצים המטפלות בקריאה וכתיבה בצורה אסינכרונית, או להשתמש ב-thread שיבצע ק/פ בזמן ש-threads אחרים בתהליך יבצעו פעילויות אחרות.

ניתן להשתמש במקשרי ההתקנים עם פונקציות אחרות המבוססות על שימוש במקשרים. יישום אינו יכול לבצע ק/פ לבתים בצורה אקראית מכיון שמחוון הק/פ מתייחס רק לקבצים. אולם, יישום יכול לשכפל מקשרי קבצים.

בדומה לק/פ בקבצים, אין מערכת הקבצים יכולה להבטיח את ביצוע בקשות הק/פ השונות לפי הסדר שהתהליך ביצע אותן. לדוגמה, אם המאגר (buffer) של היישום נמצא בסגמנט שהועבר החוצה, מערכת הקבצים תנסה לבצע בקשות ק/פ אחרות להתקן השייכות ל-threads אחרים בתהליך, ואולי היא תצליח לבצע אותם. אם תהליך תלוי בצורה קריטית במידע שנשלח להתקן, או התקבל ממנו בסדר מסוים, אין הוא צריך להתחיל ביצוע בו-זמני של מספר בקשות ק/פ, ועליו להקדיש thread אחד בלבד לביצוע כל בקשות הק/פ של התהליך, ולא להתחיל מספר threads לביצוע ק/פ לאותו התקן.

כאשר לתהליך אין עוד צורך לבצע ק/פ לקובץ, הוא סוגר את המקשר להתקן, ובכך הוא מפסיק את הקשר בינו לבין ההתקן. במקרה שהיישום מסתיים בצורה לא סדירה, מערכת ההפעלה סוגרת כל מקשר להתקן שהיה בבעלות התהליך.

המגבלה של המערכת על מספר מקשרי הקבצים כוללת גם את מקשר ההתקנים. לכן ההקצאה הבסיסית של 20 מקשרים לתהליך תופסת הן לגבי קבצים והן לגבי התקנים. תהליך יכול להגדיל את מספר המקשרים העומדים לרשותו בעזרת פונקציה של המערכת. הגדלה זו תלויה במספר התהליכים האחרים במערכת ובמספר המקשרים שהוקצו להם.

פרמטר	תיאור
שם	שם התקן התו
מצב (open FLAG)	קיים
	פתוח
אופן הגישה (open MODE)	ירושה שגיאות כשלון W/R ,W ,R DN ,DRW ,DW ,DR

אופן השיתוף
DR = Deny Readers (כתיבה בלבד)
DW = Deny Writers (קריאה בלבד)
DRW = Deny All (אין כלל הרשאות)
DN = Deny None (קריאה וכתיבה)

אופן הגישה
R = קריאה
W = כתיבה
W/R = קריאה/כתיבה

תרשים 35. הפרמטרים של DosOpen עבור התקני תו

התקנים סטנדרטיים

מספר התקני תווים מוכרים אוטומטית על-ידי מערכת הקבצים. התקנים אלה הם התקן הקלט הסטנדרטי (Standard INPUT Device), התקן הפלט הסטנדרטי (Standard OUTPUT Device) והתקן השגיאה הסטנדרטי(Standard ERROR Device). להתקנים הסטנדרטיים יש מקשרים המוגדרים מראש וכל התהליכים יכולים להשתמש בהם. מקשרים אלה הינם העתקים של מקשרים המשמשים את התקני הקונסולה (consol). תהליך יכול לקרוא מהמקלדת בעזרת מקשר קלט סטנדרטי, או לכתוב למסך הן בעזרת מקשר הפלט הסטנדרטי והן בעזרת מקשר השגיאה הסטנדרטי. ההבדל בין התקן הפלט הסטנדרטי להתקן השגיאה הסטנדרטי הוא בכך, שלא ניתן לנתב ק/פ (redirect) המיועד אל התקן השגיאה הסטנדרטי. ניתן לנתב את מקשרי הפלט והקלט הסטנדרטיים, וגם משתמש הקצה יכול לעשות זאת משורת הפקודה. אם תהליך צריך להציג למשתמש הקצה הודעה הדורשת לבצע פעולה מסוימת (ולכן לא ניתנת לניתוב לקובץ, לדוגמה), היישום צריך לכתוב את ההודעה בעזרת מקשר השגיאה הסטנדרטי.

בדומה להתקני תווים אחרים, המקשרים (handles) של ההתקנים הסטנדרטיים יכולים לשמש פונקציות מערכת המבוססות על מקשרים. פונקציות אלו כוללות את DosRead, DosWrite, DosReadAsync, DosWriteAsync, DosClose, ו-DosDupHandle.

מערכת הקבצים של OS/2 אינה תומכת בהתקן חיצוני (auxiliary) סטנדרטי, או בהתקן מדפסת סטנדרטי. התקנים אלה מוכרים על-ידי מערכת הקבצים של DOS.

צינורות

צינור הוא התקן של קובץ מדומה (pseudo-file), המאפשר לתהליכים השייכים לאותה משפחה להתקשר ביניהם בצורה הדומה לביצוע ק/פ לקובץ (ראה תקשורת בין תהליכים בפרק 3). בכל מקרה, צינור אינו התקן אחסון, אלא הוא קיים כמבנה נתונים בזיכרון.

כדי לבצע ק/פ לצינור יש צורך להשתמש בפונקציות המבוססות על מקשרים. לשם כך, הבה נערוך חזרה קטנה. בתחילה, התהליך יוצר את הצינור בעזרת פונקציית מערכת מיוחדת (DosMakePipe), אשר מחזירה שני מקשרים לצינור: אחד משמש לקריאה מהצינור והשני משמש לכתיבה לצינור. כאשר תהליך האב מתחיל את תהליך הבן, תהליך הבן יורש את המקשר המאפשר כתיבה לצינור. לאחר מכן ניתן להשתמש במקשרים אלה לביצוע ק/פ לצינור עם פונקציות לקריאה וכתיבה של מערכת הקבצים. כאשר תהליך מסיים להשתמש בצינור, הוא מנתק את החיבור לצינור על-ידי סגירת המקשר.

המקשרים לצינור הם מקשרים הנמצאים בשימושה של מערכת הקבצים, ולכן הם חלק מאותו מאגר של מקשרים שאליו שייכים גם מקשרי הקבצים ומקשרי ההתקנים. מספר התקני התווים, הצינורות והקבצים הנמצאים בשימוש של יישומים אחרים יקבע את מספר המקשרים שתהליך יוכל להשיג.

תיאור	פרמטר
דיסק/דיסקט לוגי	שם
קיים	מצב (open FLAG)
פתוח	
פתיחה ישירה - DASD	אופן הגישה (open MODE)

תרשים 36. הפרמטרים של DosOpen עבור דיסק/דיסקט לוגי

דיסק/דיסקט לוגי

דיסק/דיסקט לוגי הוא התקן המוכר ליישום כאות כונן. דיסק/דיסקט לוגי הוא שם אחר למחיצה על דיסק קשיח, או על דיסקט.

באופן רגיל, יישום פונה לדיסק/דיסקט לוגי בכל פעם שהוא מבצע ק/פ לקובץ, מכיון שקובץ נמצא על מצע מגנטי המזוהה על-ידי אות הכונן. במצבו הבסיסי, דיסק/דיסקט לוגי אינו יכול לאחסן נתונים ולכן יש צורך לפרמט אותו.

כדי לפרמט את המחיצה על הדיסק/דיסקט, חייבת תכנית השירות להכנס אליו, או לפתוח אותו, באמצעות מערכת הקבצים. השם המשמש לזיהוי ההתקן הלוגי דומה לשם קובץ, פרט לכך שהוא מציין את הכונן (תו כונן, נקודותיים) בלבד. פונקציית המערכת OPEN מחזירה מקשר מיוחד להתקן, אשר משמש את הפונקציה DosDevIOCtl (ראה הפקודות ל-IOCtl קטגוריה 8). לאחר שתכנית השירות של המערכת מפרמטת את ההתקן הלוגי, היא סוגרת את המקשר המיוחד להתקן. בתרשים 36 תמצא סיכום של הפרמטרים הדרושים לתכנית השירות כדי לפתוח דיסק/דיסקט לוגי.

מישקים לשימוש כללי

מישקים אלה משרתים את כל מגוון האובייקטים שאליהם יכולה מערכת הקבצים להתקשר, מהתקני תו ועד לצינורות.

DosClose סוגר את המקשר הספציפי (קובץ, התקן תו, צינור, התקן סטנדרטי, או דיסק/דיסקט לוגי) ומנתק את הקשר של התהליך לאובייקט. לגבי קובץ, מערכת הקבצים מעדכנת את המידע אודות הקובץ במדריך, ומנקה את המאגרים ששימשו לק/פ. לגבי התקן התו, מערכת הקבצים תודיע ל-device driver על מצב CLOSE, אם צריך.

DosDupHandle מחזיר מקשר, שהוא העתק של המקשר הספציפי שהתהליך ביקש. כדי להוריש מקשר לתהליך בן, או כדי להעביר מקשר, התהליך צריך לשכפל את המקשר שנמצא בבעלותו. ניתן לעשות זאת עם מקשרי קבצים, מקשרים לצינורות, מקשרים להתקני תו, או מקשרים להתקנים סטנדרטיים. מערכת

הקבצים יכולה ליצור מקשר חדש, או להשתמש במקשר הקיים הנמצא בבעלות התהליך. אם התהליך רוצה שמערכת הקבצים תגרום לכך שהמקשר הקיים יהיה העתק של מקשר לאובייקט הספציפי, יהיה עליה לסגור את המקשר הקיים לפני שהיא תוכל להגדיר עבורו אובייקט חדש. כל פעולת ק/פ המבוצעת עם המקשר המועתק גורמת־לעדכון מחוון הק/פ בקובץ, אשר מועבר לכל ההעתקים של אותו מקשר. מאידך, ביצוע CLOSE בעזרת מקשר מועתק משפיע רק על אותו מקשר - העתקים אחרים של המקשר ישארו פעילים.

DosOpen מחזיר מקשר לקובץ, להתקן תו, או לדיסק/דיסקט לוגי. אם האובייקט הוא קובץ, היישום מספק את הדברים הבאים:

- שם הקובץ.
- הפעולה שצריך לבצע (הפרמטר Open Flag):
  - o אם הקובץ קיים, היישום יכול לבחור בין: - כשלון הבקשה,
    - פתיחת הקובץ,
    - החלפת הקובץ.
  - o אם הקובץ אינו קיים, היישום יכול לבחור בין: - כשלון הבקשה,
    - יצירת הקובץ.
- גודל הקובץ לצורך ביצוע הפעולות CREATE (יצירת הקובץ) ו-REPLACE (החלפת הקובץ).
- מאפייני הקובץ לצורך ביצוע הפעולות CREATE ו-REPLACE (קריאה בלבד, סמוי, מערכת, מדריך או ארכיב). המאפיינים חייבים להתאים למאפיינים הנמצאים במדריך על הקובץ העומד להיות מוחלף. לא ניתן להחליף קובץ המוגדר לקריאה בלבד.
- אופן הפעולה (הפרמטר Open Mode).
  - o שימוש במאגרים של מערכת הקבצים לצורך העברת נתוני הקבצים (Write-Through Flag - כתיבה ישירה). בעת ביצוע פעולת ק/פ, המאגרים של מערכת הקבצים משמשים לאגירת נתוני הקובץ לפי הצורך. ביצוע בקשת הק/פ יכול להסתיים לפני שכל נתוני הקובץ נכתבו למצע האחסון החיצוני (media). היישום יכול להודיע למערכת הקבצים לשלוח את נתוני הקובץ אל הכונן בכל בקשת ק/פ וכך, לעקוף את העבודה במאגר. כאשר מסתיימת פעולת ק/פ, הנתונים יהיו בפועל במצע האחסון החיצוני.
  - o הטיפול בשגיאות (Fail-Errors Flag - שגיאות-כשלון). שגיאות של פעולות במצע אחסון יכולות להיות מדווחות באמצעות השיגרה לטיפול בשגיאות קריטיות של מערכת ההפעלה. הן תגרומנה להתערבות המפעיל, או שתדווחנה קוד שגיאה אל היישום, אשר יטפל בשגיאה.
  - o אפשרות להוריש את המקשר שהתקבל (Inheritance Flag - ירושה). תהליכי הבנים אינם יכולים לרשת מאפיינים מסוימים כמו: Write-Through Flag ו-Fail-Errors Flag.
  - o מטרת הפניה לקובץ (אופן הגישה - Access Mode): קריאה בלבד, כתיבה בלבד, קריאה/כתיבה.
  - o צורת השיתוף שהיישום מאפשר ליישומים אחרים (אופן השיתוף - Sharing Mode): כתיבה בלבד, קריאה בלבד, אין הרשאה כלל, מותרת קריאה וכתיבה.

אם האובייקט הוא התקן תו, היישום חייב לספק את הפרטים הבאים:

- שם התקן התו.
- סוג הפעולה כמו OPEN להתקן קיים (הפרמטר Open Flag).

- אופן הפעולה (הפרמטר Open Mode).
  - אפשרות להוריש את המקשר שהתקבל (Inheritance Flag - ירושה).
  - הטיפול בשגיאות (Fail-Errors Flag - שגיאות-כשלון).
  שגיאות יכולות להיות מדווחות באמצעות השיגרה לטיפול בשגיאות קריטיות של מערכת ההפעלה. שימוש בשיגרה זו יגרום את התערבות המפעיל. ניתן גם לדווח קוד שגיאה ישירות ליישום, אשר יטפל בשגיאה.
  - מטרת הפניה להתקן (אופן הגישה - Access Mode): קריאה בלבד, כתיבה בלבד, קריאה/כתיבה.
  - צורת השיתוף שהיישום מאפשר ליישומים אחרים (אופן השיתוף - Sharing Mode): כתיבה בלבד, קריאה בלבד, אין הרשאה כלל, מותרת קריאה וכתיבה. מערכת הקבצים כופה את אופן השיתוף כפי שנקבע על-ידי ה-device driver של ההתקן. אם ה-device driver מורה למערכת הקבצים לא לאלץ שיתוף להתקן, השיתוף לא יתקיים.

אם האובייקט הוא דיסק/דיסקט לוגי, היישום מעביר את הפרטים הבאים:

- מפרט הכונן (specification).
- סוג הפעולה, כמו OPEN להתקן קיים (הפרמטר Open Flag).
- אופן הפעולה (הפרמטר Open Mode).
  - סוג ההתקן (DASD Open Flag).

DosQFHandState מחזיר מידע על מצב המקשר, ללא תלות בסוגו. התהליך יכול לקבוע את הדברים הבאים לגבי המקשר:

- אם המקשר מייצג דיסק/דיסקט לוגי.
- אם המקשר ניתן לירושה או לא.
- אם סיום הכתיבה לקובץ מצביע על כך שהנתונים הועברו למצע מגנטי, או שהם נמצאים/לא-נמצאים במאגרים של מערכת הקבצים.
- אם שגיאות קריטיות בק/פ לקבצים מטופלות על-ידי מערכת ההפעלה, או מדווחות ישירות לתהליך.
- את סוג הגישה שניתן לבצע לאובייקט.
- את סוג השיתוף המותר לאובייקט.

DosQHandType מוסר את סוג המקשר המסוים: מקשר לקובץ, מקשר לצינור, או מקשר להתקן תו. אם המקשר הוא מקשר להתקן תו, ההתקן יזוהה כמדפסת, כמסך או כהתקן אחר. אם המקשר הוא מקשר לדיסק/דיסקט לוגי, הוא יזוהה כמקשר לקובץ.

DosRead קורא מספר מוגדר של בתים מקובץ, מצינור, או מכל התקן המזוהה על-ידי המקשר. לא יהיה ניתן לקרוא נתונים מההתקן, אם הרשאת הגישה לקובץ אינה כוללת קריאה. אם ניתן להשיג את הנתונים, התהליך חייב לבדוק את מספר הבתים שהוחזרו בפועל, מכיון שיתכן שלא היה ניתן לקבל את המספר המבוקש של הבתים. לדוגמה, אם מספר הבתים שהוחזר בפועל הוא אפס, המשמעות היא שהתהליך ניסה לקרוא מעבר לסוף הקובץ. מחוון הק/פ בקובץ מוזז למקום הבא לצורך ק/פ. כלומר, המקום החדש שווה למיקום ביצוע לפני הבקשה ועוד אורך הנתונים שהתקבלו.

DosReadAsync קריאה אסינכרונית של מספר מוגדר של בתים מקובץ, מצינור, או מכל התקן המזוהה על-ידי המקשר. התהליך מעביר אתת RAM וקוד החזרה של מאגר בנוסף לפרמטרים הרגילים של DosRead. התהליך חייב להפעיל את אתת

ה-RAM (בעזרת DosSemSet) לפני שהוא קורא ל-DosReadAsync. לאחר שמערכת הקבצים מסיימת את הפעולה ומעדכנת את מחוון הק/פ, היא מנקה את האתת. בנקודה מסוימת, התהליך ממתין (DosSemWait) שאתת RAM יהיה זמין. רק אז הוא יכול לבחון את הקוד המוחזר ואת מספר הבתים שנקראו.

DosSetFHandState קובע את המצב של המקשר, ללא תלות בסוגו. היישום יכול לקבוע את הדברים הבאים:

- את סוג המקשר שתהליך הבן יירש.
- בקשת הק/פ תחזור ליוזם לאחר שהנתונים הועברו למעשה למצע המגנטי.
- שגיאות קריטיות בעת פעולות ק/פ לקובץ יטופלו על-ידי מערכת ההפעלה, או ידווחו ישירות לתהליך.

DosSetMaxFH קובע את מספר המקשרים המקסימלי שיעמדו לרשות התהליך. כל המקשרים הפתוחים הנמצאים בבעלות התהליך אינם מושפעים על-ידי פעולה זו.

DosWrite כותב מספר מוגדר של בתים לקובץ, לצינור או לכל התקן המזוהה על-ידי המקשר. הנתונים לא יישלחו להתקן אם הקובץ מאופיין כקריאה בלבד, או שהרשאת הגישה לקובץ אינה כוללת כתיבה. התהליך חייב לבדוק את מספר הבתים שנכתבו בפועל להתקן. לגבי קבצים, אם מספר הבתים שנכתבו שונה ממספר הבתים שהיו צריכים להכתב, יתכן שאין מספיק מקום במצע המגנטי להכיל את הנתונים. מחוון הק/פ בקובץ מוזז למקום הבא לצורך ק/פ. כלומר, המקום החדש שווה למיקום לפני ביצוע הבקשה ועוד אורך הנתונים שהתקבלו. בבקשת כתיבה של נתונים הנמצאים במספר סקטורים (סקטור הוא בן 512 בתים), הסדר שבו הסקטורים יכתבו במצע האחסון אינו מובטח מראש. אם יש צורך שכמות גדולה של נתונים תכתב בסדר מסוים, היישום צריך לבצע פעולות כתיבה נפרדות ולציין את הדרישה לכתיבה ישירה (Write-Through Flag Set). בדרך זו הנתונים יכתבו ישירות למצע המגנטי לפני שהפיקוח יחזור לתהליך.

DosWriteAsync כתיבה אסינכרונית של מספר מוגדר של בתים אל קובץ, צינור, או אל כל התקן המזוהה על-ידי המקשר. התהליך מעביר אתת RAM וקוד החזרה של מאגר בנוסף לפרמטרים הרגילים של DosWrite. התהליך חייב לאתחל את אתת ה-RAM (DosSemSet) לפני שהוא קורא ל-DosWriteAsync. לאחר שמערכת הקבצים מסיימת את הפעולה ומעדכנת את מחוון הק/פ, היא מנקה את האתת. בנקודה מסוימת, התהליך ממתין (DosSemWait) שאתת RAM יהיה זמין. רק אז הוא יכול לבחון את הקוד המוחזר ואת מספר הבתים שנכתבו.

מישקים המבוססים על קבצים

סיכמנו בשתי קטגוריות את מישקי מערכת הקבצים שתוכננו במיוחד לקבצים: מישקים המבוססים על שם הקובץ, ומישקים המבוססים על המקשר לקובץ. המישקים המבוססים על שם הקובץ דורשים מחרוזת ASCII המסתיימת בבית 00h (ידוע גם בשם ASCIIZ). עבור המישקים המבוססים על מקשרים דרוש מקשר לקובץ.

הערה: הסימון nnh או nnnnh בהמשך מציין תווים בקוד הקסה-דצימלי (h).

ניתן להשתמש במישקים אלה עם קבצים, מבלי להתחשב בכך שהם נפתחו או לא. אולם, מספר פעולות דורשות שהקובץ לא יהיה בשימוש.

DosDelete מוחק קובץ מסוים במסלול חיפוש מוגדר. אם לא הוגדר מסלול חיפוש, הוא מוחק את הקובץ מהמדריך הנוכחי. קבצים עם המאפיין של קריאה בלבד אינם ניתנים למחיקה. אם רוצים למחוק קובץ כזה, חייבים לשנות תחילה את מאפיין הקובץ. גם קבצים שנפתחו על-ידי תהליכים אחרים אינם ניתנים למחיקה, מכיון שהפונקציה DELETE פועלת כמשתמש בלעדי של הקובץ.

DosFindClose סוגר מפתח חיפוש למדריך, כדי להפסיק חיפוש של שם קובץ או קבוצה של שמות קבצים. הפונקציה DosFindFirst יכולה להתחיל חיפוש במדריך.

DosFindFirst מוצא את הקובץ הראשון ששמו שווה לשם הקובץ שמחפשים במדריך שהוגדר. אם לא הוגדר מדריך (directory) מסוים, הוא מחפש את הקובץ במדריך הנוכחי. לאחר מכן הוא מחזיר את מפתח החיפוש בקובץ כדי לאפשר ל-DosFindNext להמשיך את החפוש במדריך זה. התהליך יכול לציין מפתח של מדריך מחדל (default directory) (0001h). ניתן לחפש תמיד במדריך זה, או לבקש שהמפתח של המדריך יוחזר. אם התהליך מציין מפתח של מדריך שמחפשים בו באותו רגע, המפתח נסגר והבקשה מצורפת לבקשה הנוכחית. התהליך יכול לציין שם קובץ שיכול להכיל תווים גלובליים ומאפיינים (נורמלי, סמוי וכו'). התוצאות של החיפוש יועברו למאגר שצוין על-ידי היישום.

DosFindNext מוצא את הקובץ הבא ששווה לשם הקובץ ולמדריך כפי שצוינו במפתח החיפוש. מפתח זה מוחזר על-ידי הפונקציה DosFindFirst. התהליך משתמש ב-DosFindNext עד אשר אין יותר קבצים אשר שווים למפתח החיפוש. המידע על הקבצים האלה מועבר למאגר שצוין על-ידי היישום.

DosMove מעביר קובץ למדריך ו/או לקובץ אחר. התהליך מציין את שם המסלול/קובץ הנוכחי ואת שם מסלול/קובץ המטרה. אם משתמשים באות כונן, הכונן חייב להיות זהה הן בקבצי המטרה והן בקבצי המקור. זאת אומרת, שניתן להעביר קובץ למדריך אחר באותו כונן ולתת לו שם אחר, אבל לא ניתן להעבירו לכונן אחר.

DosQFileMode מחזיר את המאפיינים שנקבעו לקובץ שזוהה על פי שמו. המאפיינים (attributes) הם: קריאה בלבד, סמוי, מערכת, תת-מדריך, וארכיב.

DosSearchPath מחפש שם קובץ מסוים במסלולי החיפוש שצוינו. היישום קובע את סדר החיפוש באחת מהדרכים הבאות:

- מחרוזת ASCII המכילה מספר מסלולי חיפוש המופרדים על-ידי נקודה-פסיק ומסתיימת בבית שערכו 00h.
- מחרוזת ASCII המכילה שם של משתנה המכיל מסלול חיפוש, הנמצא בסביבה של התהליך.

החיפוש במסלולים השונים נעשה לפי הסדר שצוין במחרוזת. אם נמצא שם של קובץ התואם את השם שצוין שם, המדריך ושם הכונן מועברים למאגר שצוין

ניתן להשתמש במישקים אלה עם קבצים, מבלי להתחשב בכך שהם נפתחו או לא. אולם, מספר פעולות דורשות שהקובץ לא יהיה בשימוש.

DosDelete מוחק קובץ מסוים במסלול חיפוש מוגדר. אם לא הוגדר מסלול חיפוש, הוא מוחק את הקובץ מהמדריך הנוכחי. קבצים עם המאפיין של קריאה בלבד אינם ניתנים למחיקה. אם רוצים למחוק קובץ כזה, חייבים לשנות תחילה את מאפיין הקובץ. גם קבצים שנפתחו על-ידי תהליכים אחרים אינם ניתנים למחיקה, מכיון שהפונקציה DELETE פועלת כמשתמש בלעדי של הקובץ.

DosFindClose סוגר מפתח חיפוש למדריך, כדי להפסיק חיפוש של שם קובץ או קבוצה של שמות קבצים. הפונקציה DosFindFirst יכולה להתחיל חיפוש במדריך.

DosFindFirst מוצא את הקובץ הראשון ששמו שווה לשם הקובץ שמחפשים במדריך שהוגדר. אם לא הוגדר מדריך (directory) מסוים, הוא מחפש את הקובץ במדריך הנוכחי. לאחר מכן הוא מחזיר את מפתח החיפוש בקובץ כדי לאפשר ל-DosFindNext להמשיך את החפוש במדריך זה. התהליך יכול לציין מפתח של מדריך מחדל (default directory) (0001h). ניתן לחפש תמיד במדריך זה, או לבקש שהמפתח של המדריך יוחזר. אם התהליך מציין מפתח של מדריך שמחפשים בו באותו רגע, המפתח נסגר והבקשה מצורפת לבקשה הנוכחית. התהליך יכול לציין שם קובץ שיכול להכיל תווים גלובליים ומאפיינים (נורמלי, סמוי וכו'). התוצאות של החיפוש יועברו למאגר שצוין על-ידי היישום.

DosFindNext מוצא את הקובץ הבא ששווה לשם הקובץ ולמדריך כפי שצוינו במפתח החיפוש. מפתח זה מוחזר על-ידי הפונקציה DosFindFirst. התהליך משתמש ב-DosFindNext עד אשר אין יותר קבצים אשר שווים למפתח החיפוש. המידע על הקבצים האלה מועבר למאגר שצוין על-ידי היישום.

DosMove מעביר קובץ למדריך ו/או לקובץ אחר. התהליך מציין את שם המסלול/קובץ הנוכחי ואת שם מסלול/קובץ המטרה. אם משתמשים באות כונן, הכונן חייב להיות זהה הן בקבצי המטרה והן בקבצי המקור. זאת אומרת, שניתן להעביר קובץ למדריך אחר באותו כונן ולתת לו שם אחר, אבל לא ניתן להעבירו לכונן אחר.

DosQFileMode מחזיר את המאפיינים שנקבעו לקובץ שזוהה על פי שמו. המאפיינים (attributes) הם: קריאה בלבד, סמוי, מערכת, תת-מדריך, וארכיב.

DosSearchPath מחפש שם קובץ מסוים במסלולי החיפוש שצוינו. היישום קובע את סדר החיפוש באחת מהדרכים הבאות:

- מחרוזת ASCII המכילה מספר מסלולי חיפוש המופרדים על-ידי נקודה-פסיק ומסתיימת בבית שערכו 00h.
- מחרוזת ASCII המכילה שם של משתנה המכיל מסלול חיפוש, הנמצא בסביבה של התהליך.

החיפוש במסלולים השונים נעשה לפי הסדר שצוין במחרוזת. אם נמצא שם של קובץ התואם את השם שצוין שם, המדריך ושם הכונן מועברים למאגר שצוין

על-ידי היישום. שם הקובץ שצוין מתוסף למסלול הזה, אפילו אם שם הקובץ הכיל תווים גלובליים. הדבר מאפשר ליישום למצוא מדריך המכיל שם קובץ אחד או יותר, התואם לשם הקובץ שצוין בבקשה, מבלי לשנות את ה-DosFindFirst.

DosSetFileMode קובע את המאפיינים של קובץ שזוהה לפי שם הקובץ. מאפייני הקובץ הניתנים לשינוי הם: קריאה בלבד, סמוי, מערכת וארכיב.

מישקים המבוססים על מקשרים לקבצים

מישקים אלה דורשים שקובץ הקלט יפתח קודם שהם מופעלים.

DosBufReset מנקה את המאגרים של מערכת הקבצים ששימשו את המקשר לקובץ שצוין. מערכת הקבצים מעדכנת את המידע על הקובץ במדריך באותה דרך כפי שהיא היתה עושה אילו הקובץ היה נסגר, אולם הקובץ נשאר פתוח.

התהליך יכול לציין מקשר גלובלי (FFFFh) לקובץ, אשר ינקה את כל המאגרים של מערכת הקבצים הקשורים לכל הקבצים הפתוחים של התהליך.

DosChgFilePtr מזיז את מחוון הק/פ בקובץ עבור מקשר הקובץ המסוים. תכנית היישום מציינת הן את כיוון התנועה והן את המרחק. ניתן להזיז את מחוון הק/פ כדלקמן:

- מתחילת הקובץ פלוס המרחק שצוין.
- מהמקום הנוכחי של מחוון הק/פ פלוס המרחק שצוין.
- מסוף הקובץ (EOF) פלוס המרחק שצוין.

DosFileLocks נועל לשימוש בלעדי ומשחרר טווח של בתים בקובץ פתוח המזוהה על פי המקשר לקובץ. התהליך מציין את איזור המטרה ככתובת יחסית מתחילת הקובץ, ואת גודל השטח שיושפע מכך. התהליך יכול לציין בפקודה אחת, כפרמטרים של הפונקציה, את האיזור שיש לנעול ואת האיזור שיש לשחרר. כאשר יש צורך לשחרר ולנעול בפקודה אחת, מערכת הקבצים תשחרר לפני שתנעל. אם נועלים איזור העובר את סוף הקובץ לא תופק הודעת שגיאה.

מנגנון הנעילה ושחרור של טווח מונע באופן זמני מתהליכים אחרים מלהכנס לשטחים בקובץ הפתוח. תהליך ישתמש בנעילה כדי למנוע מתהליכים אחרים מלכתוב בו-זמנית לאיזור מסוים בקובץ. הוא יכול גם למנוע באופן זמני מתהליכים אחרים מלקרוא איזור מסוים בקובץ בזמן שהוא כותב לאיזור זה.

אם קובץ נסגר כאשר הנעילה עדיין בתוקף, מערכת הקבצים משחררת את הנעילה, אך היא אינה עושה זאת בסדר מוגדר כלשהו. אם מעתיקים מקשר לקובץ שיש לו גישה לאיזור הנעול, גם לעותק תהיה גישה לאיזור זה. אולם, אם תהליך-בן צריך לרשת מקשר לקובץ שיש לו גישה לאיזור הנעול, המקשר הזה לא יוכל לגשת לאותו איזור.

DosNewSize משנה את הגודל של קובץ המזוהה על-ידי המקשר שלו. התהליך מציין את הגודל החדש של הקובץ, שמציין אם חל גידול או הקטנה. שינוי זה לא ישתקף במידע על הקובץ עד אשר הוא יסגר. ההקצאה של השטח הנוסף לקובץ נעשית, עד כמה שאפשר, מתוך שטח הסמוך לשטח שהקובץ נמצא בו כרגע. שינוי בגודל הקובץ אין פירושו העתקה פיסית של הקובץ למקום חדש. אי אפשר לציין גודל חדש לקובץ המוגדר לקריאה בלבד.

DosQFileInfo מחזיר מידע על קובץ שזוהה בעזרת המקשר שלו. התהליך יכול לקבל את הפרטים הבאים על הקובץ:

- תאריך וזמן של יצירת הקובץ.
- תאריך וזמן של הגישה האחרונה לקובץ.
- תאריך וזמן של הכתיבה האחרונה לקובץ.
- כמות הנתונים בבתים.
- כמות ההקצאה הכוללת לקובץ בבתים.
- מאפיין.

DosSetFileInfo קובע פריטי מידע מסוימים אודות קובץ המזוהה על-ידי המקשר שלו. ניתן לעשות זאת רק לגבי קבצים שנפתחו לצורך כתיבה בלבד. התהליך יכול לשנות את פריטי המידע הבאים:

- תאריך וזמן של יצירת הקובץ.
- תאריך וזמן של הגישה האחרונה לקובץ.
- תאריך וזמן של הכתיבה האחרונה לקובץ.

מישקים המבוססים על המדריך

מישקים אלה מאפשרים לתהליך לטפל במדריכים.

DosChDir משנה את המסלול הנוכחי כפי שצוין בבקשה. המסלול אינו משתנה אם חלק כלשהו של המסלול שצוין בבקשה אינו קיים.

DosMkDir יוצר תת-מדריך כפי שצוין בבקשה. תת-המדריך לא יווצר אם חלק כלשהו של המסלול שצוין בבקשה לא קיים.

DosQCurDir מחזיר את שם המסלול הנוכחי בהתאם לכונן שצוין בבקשה.

DosQCurDisk מחזיר את כונן ברירת המחדל ומפת סיביות, שבה מצוין אילו מבין 26 אותיות הכונן האפשריות מתאימות לדיסק/דיסקט הלוגי בזמן הבקשה.

DosRmDir מבטל תת-מדריך המזוהה על-ידי המסלול שצוין בבקשה. כמו ב-DOS, לא ניתן לבטל תת-מדריך אם הוא מכיל קבצים כלשהם, כולל קבצים סמויים. כמו כן, לא ניתן לבטל תת-מדריך אם הוא המדריך הנוכחי, או שהוא המדריך הראשי.

מישקים להתקנים המאחסנים קבצים

מישקים אלה משמשים לטיפול במידע על התקנים המאחסנים קבצים.

DosQFsInfo מחזיר את פרטי המידע של מערכת הקבצים הקיימים ברמת המידע שצוינה בבקשה. המידע המוחזר הוא על דיסק/דיסקט לוגי המזוהה על-ידי מספר הכונן. רמת המידע הראשונה של מערכת הקבצים מכילה את הפרטים הבאים:

- מספר סקטורים ביחס ליחידת הקצאה.
- מספר יחידות ההקצאה.
- מספר יחידות ההקצאה החופשיות.
- מספר הבתים בסקטור.

רמת המידע השניה של מערכת הקבצים מתייחסת לתווית השם של הכונן:

- תאריך וזמן יצירה.
- אורך המחרוזת של תווית השם.
- מחרוזת ASCII המכילה את תווית השם של הכונן.

DosQVerify מחזיר את המצב הנוכחי של אימות תוך כדי כתיבה לקובץ (write with verify), או להתקן אחר. התהליך משתמש בנוהל זה כדי להבטיח שנתונים קריטיים ייכתבו להתקן ללא שגיאה.

DosSelectDisk בוחר את הכונן שצוין בבקשה ככונן ברירת המחדל של התהליך שביקש.

DosSetFsInfo קובע את רמת המידע שמערכת הקבצים תציג לגבי דיסק/דיסקט לוגי המזוהה על-ידי מספר הכונן. רמת המידע הראשונה שמערכת הקבצים מציגה מתייחסת להתקן לוגי ולא ניתנת לשינוי על-ידי התהליך. הרמה השניה של המידע מתייחסת לתווית השם של הכונן. היישום יכול לשנות את המחרוזת המכילה את התווית של הכרך (volume) ואת אורכה רק כאשר הכרך נפתח למטרת כתיבה.

DosSetVerify קובע את המצב של אימות נתונים תוך כדי כתיבה לכל הקבצים ולכל ההתקנים שאליהם מבצע התהליך ק/פ. התהליך משתמש במצב זה כדי להבטיח שמידע קריטי נכתב להתקן ללא שגיאות.

תת-מערכות לביצוע ק/פ להתקני תו

תת-מערכת ב-OS/2 היא קבוצה של שירותים המכוונים לטפל בהתקן, או במאפיין מסוים. תת-מערכת לק/פ יכולה לקבל צורות שונות, אשר ביניהן יש 3 צורות עיקריות:

- **device driver**, אשר תומך בק/פ של היישום להתקן שלו בעזרת מישקים רגילים של מערכת ההפעלה.
- **ספריית הקישור הדינמי**, אשר תומכת בק/פ של היישום בעזרת מישקים שלו שהותאמו לטיפול בק/פ.
- **תהליך**, אשר תומך בק/פ של תהליכים אחרים, באמצעות מישק המבוסס על תקשורת בין תהליכים (IPC).

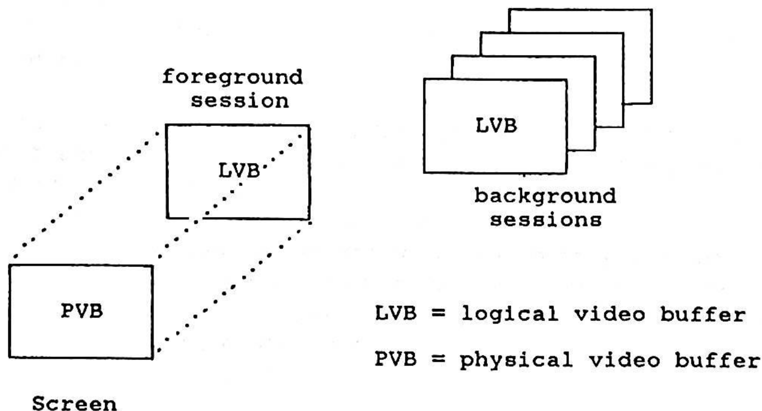
תת-המערכת יכולה להכיל שילוב של שלוש האפשרויות.
במהדורה הסטנדרטית של OS/2 ניתן למצוא 3 תת-מערכות לק/פ המשרתות את התקני הקונסולה: מסך, מקלדת ועכבר. כל אחת מתת-מערכות אלו מתבססת על ספריית קישור דינמי של מישקים ליישום ועל device driver. שמות תת-המערכות הם:

- VIO, תת-מערכת לתצוגה.
- KBD, תת-מערכת למקלדת.
- MOU, תת-מערכת לעכבר.

VIO: ק/פ לתצוגה

תת-המערכת VIO מתבססת על ספריית קישור דינמי ו-device driver. המישקים השייכים לספריית הקישור מתחילים עם הקידומת "VIO", וה-device driver הוא חלק ממערכת ההפעלה. מישקי ה-VIO מספקים פונקציות ייעודיות לטקסט, אשר מהוות קבוצת-על של הפונקציות המסופקות על-ידי מישקי התצוגה של ה-BIOS בסביבת ה-DOS. תת-המערכת VIO מספקת שתי שיטות בסיסיות לביצוע הק/פ למסך: באמצעות מאגר תצוגה לוגי (Logicl Video Buffer - LVB), או באמצעות מאגר תצוגה פיסי (Physical Video Buffer - PVB).

כפי שתזכור, בכל פעם ש-session מתחיל, מערכת ההפעלה יוצרת לו מאגר תצוגה לוגי, אשר מתחלק בין כל התהליכים המתבצעים באותו session.. לכל session יש מאגר תצוגה לוגי משלו. מאגר התצוגה הלוגי נקשר למאגר תצוגה פיסי (המסך) בכל פעם שה-session עובר לחזית (foreground) כתוצאה מהוראה של משתמש הקצה (ראה תרשים 37). בכל פעם שמשתמש הקצה עובר מ-session אחד למשנהו, תת-המערכת VIO שומרת את המסך של ה-session הקודם ומציגה את המסך של ה-session החדש. מאגר התצוגה הלוגי של ה-session הנמצא ברקע (background), מאפשר לתהליך הנמצא בו להמשיך את הק/פ לתצוגה; אולם, מאגר התצוגה הלוגי לא יוצג, עד אשר ה-session יועבר לחזית. ישנו מקרה אחד יוצא דופן, שבו תהליך הנמצא ברקע מציג בצורה זמנית מידע בחזית: תהליך זה יכול להשתמש ב-foreground session מיוחד הנקרא "pop-up" (מופיע שלא בסדר הרגיל), כדי להציג מידע על אירועים קריטיים ולחכות לתשובה ממשתמש הקצה.



תרשים 37. תצוגה בחזית מול תצוגה ברקע

תהליך יכול לפנות למאגר התצוגה הפיסי באמצעות תת-המערכת VIO והוא אינו חייב להשתמש במאגר התצוגה הלוגי לצורך ק/פ. התהליך המטפל ישירות במאגר התצוגה הפיסי חייב להיות כפוף למספר כללים, מכיון שמאגר התצוגה הפיסי חייב להתחלק בין ה-sessions. התהליך חייב לנעול ולשחרר את מאגר התצוגה הפיסי כדי שהק/פ שלו יוכל להיות מתואם עם ה-sessions האחרים, הן בזמן שה-session נמצא בחזית, והן בזמן שה-session שלו נמצא ברקע. התהליך חייב גם להיות מסוגל לקבל הודעות על מצב ה-session שלו כאשר יש צורך בביצוע אסינכרוני של sessions. כאשר משתמש הקצה עובר מה-session שבו נמצא התהליך, התהליך מקבל הודעה לשמור את המסך שלו; כאשר המשתמש חוזר ל-session, התהליך מקבל הודעה לשחזר את המסך.

הבחירה באיזה סוג מאגר להשתמש לצורך הק/פ תלויה בסוג התצוגה הרצוי. רוב מישקי ה-VIO מיועדים לטפל בטקסט, ובמקרה זה היישום יכול להשתמש במאגר התצוגה הלוגי וגם לטפל בו ישירות. אולם אם היישום צריך להשתמש במצב הגרפי, הוא חייב לטפל ישירות במאגר התצוגה הפיסי. במקרה זה ביצוע מהיר של הק/פ אינו הבעיה הקריטית, כי מישקי ה-VIO מתוכננים להגיב במהירות, כדי שהיישומים הפועלים בסביבת ביצוע רבת משימות לא יצטרכו לטפל באופן ישיר בחומרה. השימוש במאגר התצוגה הלוגי הוא בסיס לנתיב הגירה לסביבת החלונות והשירותים הגרפיים של מנהל התצוגה המסופק עם מערכת ההפעלה OS/2 גירסה 1.1.

מישקי ה-VIO מקובצים במספר קטגוריות כלליות:

- ק/פ של תווים.
- בקרת הסמן.
- הניהול של גלילת המסך (scroll).
- הבקרה על התקן התצוגה.
- הניהול של תצוגה שאינה מופיעה בסדר הרגיל (pop-up).
- ניהול מאגר התצוגה הלוגי.
- ניהול מאגר התצוגה הפיסי.
- החלפת פונקציות.

VIO ק/פ של תווים

כאשר אומרים שיישום מבצע ק/פ למסך, המשמעות היא שהוא עושה זאת באמצעות תווים והמאפיינים שלהם (צבע, עוצמה וכו'). תו המשולב עם המאפיין שלו נקרא יחידת תו (character cell).

VioReadCellStr קורא מחרוזת של יחידות תו המתחילה ממקום מסוים במאגר התצוגה, כפי שצוין. אם הקריאה מגיעה לסוף השורה והמחרוזת לא הסתיימה, הקריאה תמשך בשורה הבאה. אם הקריאה מגיעה לסוף המסך והמחרוזת לא הסתיימה, הקריאה מסתיימת ואורך המחרוזת הוא מספר הבתים הנמצא במאגר של היישום.

VioReadCharStr קורא מחרוזת של תווים המתחילה ממקום מסוים במאגר התצוגה. אם מגיעים לסוף השורה והמחרוזת לא הסתיימה, הקריאה תמשך בשורה הבאה. אם מגיעים לסוף המסך והמחרוזת לא הסתיימה, הקריאה מסתיימת ואורך המחרוזת הוא מספר הבתים הנמצא במאגר של היישום.

VioWrtCellStr כותב מחרוזת של יחידות תו המתחילה במקום מסוים במאגר התצוגה. אם מגיעים לסוף השורה והמחרוזת לא הסתיימה, הכתיבה תמשך בשורה הבאה. אם מגיעים לסוף המסך והמחרוזת לא הסתיימה, הכתיבה מסתיימת.

VioWrtCharStr כותב מחרוזת של תווים המתחילה במקום מסוים במאגר התצוגה. אם מגיעים לסוף השורה והמחרוזת לא הסתיימה, הכתיבה תמשך בשורה הבאה. אם מגיעים לסוף המסך והמחרוזת לא הסתיימה, הכתיבה מסתיימת.

VioWrtCharStrAtt כותב מחרוזת של תווים המתחילה במקום מסוים במאגר התצוגה תוך שימוש במאפיין כללי לכל התווים. אם מגיעים לסוף השורר והמחרוזת לא הסתיימה, הכתיבה תמשך בשורה הבאה. אם מגיעים לסוף המסך והמחרוזת לא הסתיימה, הכתיבה מסתיימת.

VioWrtNAttr כותב N פעמים את המאפיין ממקום מסוים במאגר התצוגה. אם מגיעים לסוף השורה והמחרוזת לא הסתיימה, הכתיבה תמשך בשורה הבאה. אם מגיעים לסוף המסך והמחרוזת לא הסתיימה, פעולת הכתיבה מסתיימת.

VioWrtNCell כותב N פעמים את יחידת-התו ממקום מסוים במאגר התצוגה. אם מגיעים לסוף השורה והמחרוזת לא הסתיימה, הכתיבה תמשך בשורה הבאה. אם מגיעים לסוף המסך והמחרוזת לא הסתיימה, פעולת הכתיבה מסתיימת.

VioWrtNChar כותב N פעמים את התו ממקום מסוים במאגר התצוגה. אם מגיעים לסוף השורה והמחרוזת לא הסתיימה, הכתיבה תמשך בשורה הבאה. אם מגיעים לסוף המסך והמחרוזת לא הסתיימה, פעולת הכתיבה מסתיימת.

VioWrtTTy כותב מחרוזת תווים במאגר התצוגה המתחילה במקום הנוכחי של הסמן ומזיז אותו למקום אחד אחרי סוף המחרוזת. אם מגיעים לסוף השורה והמחרוזת לא הסתיימה, הכתיבה תמשך בשורה הבאה. אם מגיעים לסוף המסך והמחרוזת לא הסתיימה, מתבצעת גלילה של המסך ופעולת הכתיבה ממשיכה.

תווים מסוימים מטופלים כפקודות: התו CarriageReturn, התו LineFeed, התו BackSpace, התו Tab ותו הזמזם (Bell). הטיפול בתווים מושפע על-ידי ANSI, כלומר אם ANSI הופעל. שיטת טיפול זו בתווי התצוגה ידועה בשם TTY.

VIO בקרה על הסמן

יישום יכול לשלוט במיקום ובצורה של הסמן, אשר משמש בדרך כלל למיקום תווים המוזנים במקלדת.

VioGetCurPos מחזיר את השורה והטור הנוכחיים שבהם נמצא הסמן.

VioGetCurType מחזיר את הגובה, הרוחב והמאפיין הנוכחיים של הסמן.

VioSetCurPos קובע את מיקום הסמן לפי השורה והטור שצוינו בבקשה.

VioSetCurType קובע את הגובה, הרוחב והמאפיין של הסמן.

VIO ניהול של גלילת המסך

היישום יכול להשתמש בגלילה של שטחים מרובעים בתוך המסך ובתוך מאגר התצוגה.

VioScrollDn מגולל כלפי מטה שורות טקסט בתוך שטח מרובע, עד לסיום מספר השורות שצוין בבקשה. הוא גם מכניס את מספר השורות שנגללו בעזרת יחידת תו המוצגת בראש השטח שבו בוצעה הגלילה.

VioScrollLf מגולל לכיוון שמאל שורות טקסט בתוך שטח מרובע. התזוזה תהיה לפי מספר הטורים שצוין. הוא גם מציג את מספר הטורים שנגללו בצד ימין של השטח שבו בוצעה הגלילה.

VioScrollRt מגולל לכיוון ימין שורות טקסט בתוך שטח מרובע. התזוזה תהיה לפי מספר הטורים שצוין. הוא גם מציג את מספר הטורים שנגללו בצד שמאל של השטח שבו בוצעה הגלילה.

VioScrollUp מגולל כלפי מעלה שורות טקסט בתוך שטח מרובע עד לסיום מספר השורות שצוין. הוא גם מכניס את מספר השורות שנגללו בעזרת יחידת תו המוצגת בתחתית השטח שבו בוצעה הגלילה.

VIO בקרה על התקן התצוגה

יישום יכול להשתמש בסוגי המסכים והמתאמים השונים בעזרת מגוון של מישקים. מישקים אלה מאפשרים לקבל ולקבוע את מצבי התצוגה, המידע על ההתקן ודפי קוד (או סוג האותיות המופיע במסך). בנוסף לכך, היישום יכול להפעיל בקרה מורחבת על המסך והמקלדת כפי שמוגדרת על-ידי ANSI, המווה חלק מתת-המערכת VIO (ב-DOS, הבקרה המורחבת על המסך והמקלדת לפי ANSI דרשו לציין ב-CONFIG.SYS את ה-device driver ששמו ANSI.SYS). מצב ANSI מגדיר צירופי תווים הממקמים את הסמן, מוחקים טקסט מהמסך, קובעים את מצב הק/פ למסך, ומגדירים מחדש את המשמעות של תווי המקלדת.

VioGetAnsi מחזיר את המצב הנוכחי של הבקרה המורחבת על המסך והמקלדת לפי ANSI.

VioGetConfig מחזיר את סוג המסך, סוג מתאם התצוגה ואת כמות הזיכרון הנמצאת במתאם התצוגה. הזיהוי של המסך ושל מתאם התצוגה מבוסס על מספר בדיקות שמבוצעות על-ידי תת-המערכת VIO, שמטרתן לקבוע את הקונפיגורציה של אמצעי התצוגה. אין זה אפשרי לזהות בוודאות את הקונפיגורציה של אמצעי התצוגה, כי יתכן שמצב מתגי התצוגה אינו תואם את המצב בפועל והתצוגה אינה קשורה למעשה למתאם התצוגה. פרטי המידע החוזרים משקפים את ההשערות של תת-המערכת VIO לגבי מצב העבודה בפועל.

VioGetCp מחזיר את מספר הזיהוי של דף הקוד (גופן - font) הנוכחי.

VioGetFont מחזיר את הגופן (דף קוד) הנוכחי, או גופן מסוים הנמצא ב-ROM של מתאם התצוגה (בתנאי שהמתאם מטפל בפונטים).

VioGetMode מחזיר את מצב התצוגה במונחים של מצב עבודה, מספר הצבעים, מספר השורות והטורים להצגת תווים ואת הרזולוציה האופקית והאנכית. מצבי

העבודה השונים כוללים מצב תואם-מונוכרום, מצב טקסט, מצב גרפי, ושילוב צבעים, או ציון שאין צבעים. מספר הצבעים ניתן בחזקות של 2; לדוגמה, 1 מצביע על 2 צבעים ו-2 מצביע על 4 צבעים. מספר השורות והטורים מצביע על הרזולוציה (במונחים של תווים) במצב טקסט. הרזולוציה האופקית והאנכית מצביעה על מספר הפיקסלים (pels) במסך.

VioGetState מחזיר את המצב הנוכחי של אוגרי טבלת הצבעים, של צבעי קווי הגבול, ושל עוצמת ההבהוב והרקע.

VioSetAnsi מפעיל ומפסיק את הבקרה המורחבת של המסך והמדפסת לפי ANSI.

VioSetCp קובע את דף הקוד, או את הגופן, לדף המזוהה על פי מספר הזיהוי של דף הקוד.

VioSetFont מטעין את סוג האותיות, או את דף הקוד, לתוך מתאם התצוגה. סוג האותיות חייב להיות תואם עם מצב התצוגה הנוכחי, וניתן להשתמש בו רק באותם מתאמי תצוגה התומכים בשינוי של סוגי אותיות.

VioSetMode קובע את מצב התצוגה. המצב נקבע במונחים של סוג, צבעים, מספר שורות וטורים לטקסט, ורזולוציה אופקית ואנכית. סוגי המצבים כוללים מצב תואם מונוכרום, מצב טקסט, מצב גרפי, ושילוב צבעים או ללא צבעים. מספר הצבעים צריך להיות רשום בחזקות של 2; לדוגמה, 1 מצביע על 2 צבעים ו-2 מצביע על 4 צבעים. מספר השורות והטורים מצביע על הרזולוציה (במונחים של תווים) במצב טקסט. הרזולוציה האופקית והאנכית מצביעה על מספר הפיקסלים האפשריים במסך. קביעת המצב מתחלת את סוג הסמן ומיקומו.

VioSetState קובע את המצב של אוגרי טבלת הצבעים, של צבעי קווי הגבול, ושל עוצמת ההבהוב והרקע.

VIO תצוגה שלא בסדר הרגיל (Pop-up)

תהליך המופעל ברקע יכול להכניס חלון זמני, חלון צץ או "pop-up", לתמונת המסך של ה-foreground session. הדבר מאפשר לתהליך הנמצא ברקע להודיע למשתמש על אירוע חשוב ולאפשר לו לפעול בהתאם.

VioEndPopUp מבטל את הבעלות על מאגר התצוגה הזמני בחזית.

VioPopUp מבקש מאגר תצוגה זמני בחזית. במידה ואין מאגר פנוי, נקבל קוד שגיאה, או שהתהליך ימתין עד אשר מאגר יתפנה, בהתאם לפעולה שהתהליך ביקש לבצע בחזית. אם הקוד החוזר מביצוע פונקציה זו מצביע על הצלחה, התהליך יוכל להשתמש בהתקני הקונסולה (מסך, מדפסת ועכבר) לצורך ק/פ. כל שירותי ה-VIO שהתהליך מבקש יופנו מרגע זה למאגר (buffer) התצוגה הזמני בחזית, עד אשר התהליך יקרא ל-VioEndPopUp. רק יישום אחד יכול לבצע pop-up בו-זמנית, ולכן יתכן שהיישום המבקש יצטרך להמתין עד אשר תורו יגיע. בזמן שה-"pop-up" מתבצע, אין המשתמש יכול להחליף באופן זמני sessions, ותהליכים השייכים ל-foreground session אינם יכולים לבצע את הפונקציות שלהם הקשורות במסך. אולם, אם התהליך שהוא הבעלים של ה-pop-up נמצא ב-foreground session, אז רק תהליכים השייכים ל-foreground session אינם יכולים לבצע את הפונקציות שלהם הקשורות

במסך. בנוסף לכך, כאשר ה-pop-up מתבצע, אין בעליו יכול להכנס למאגר התצוגה הפיסי.

VIO ניהול של מאגר התצוגה הלוגי

יישום יכול לטפל ישירות במאגר התצוגה הלוגי, מבלי לפגוע בתמונת המסך המוצגת באותו זמן. הוא גם יכול לגרום לכך שמאגר התצוגה הלוגי יעדכן את מאגר התצוגה הפיסי.

VioGetBuf משיג אפשרות מיעון (ערך בורר וכתובת יחסית) ל-LDT (טבלת מתארים מקומית) של מאגר התצוגה הלוגי. לשם כך הוא מקבל את הגודל של מאגר התצוגה הלוגי, אשר תלוי במצב התצוגה הנוכחי.

VioShowBuf מעדכן את מאגר התצוגה הפיסי עם תוכן מאגר התצוגה הלוגי. מישק זה יעבוד בתנאי שהתהליך המשתמש בו נמצא ב-foreground session. למישק זה אין השפעה אם התהליך נמצא ב-background session.

VIO ניהול של מאגר התצוגה הפיסי

המערכת OS/2 גירסה 1.0 אינה מספקת בצורה ישירה פונקציות התומכות במצבי התצוגה הגרפיים. לכן היא מספקת ליישום שירותי תצוגה, המאפשרים לו לבצע עבודות בגרפיקה וגם לפעול עם יישומים המתבצעים ב-sessions אחרים.

הפרוטוקול לביצוע גרפיקה במסך הוא כדלקמן:

1. השג אפשרות מיעון למאגר התצוגה הפיסי.
2. קבע מצב תצוגה גרפי (בעזרת VIOSetMode).
3. נעל את מאגר התצוגה הפיסי לצורך ביצוע ק/פ.
4. בצע ק/פ למאגר התצוגה הפיסי.
5. שחרר את מאגר התצוגה הפיסי.

יישום גרפי צריך להקדיש thread אחד לביצוע שמירה ושחזור של המסך שלו, מכיון שהמפעיל יכול לעבור מ-session ל-session. היישום יצטרך להקדיש thread נוסף לביצוע שמירה של מצב התצוגה ושל תוכן האוגרים של מתאם התצוגה, אם הוא גם מטפל בצורה ישירה באוגרים של מתאם התצוגה.

VioGetPhysBuf משיג אפשרות מיעון ל-LDT של מאגר התצוגה הפיסי. התהליך מזהה את מאגר התצוגה הזה על-ידי ציון כתובתו, אשר מכילה כתובת פיסית בת 32 סיביות וערך נוסף המציין את גודל המאגר. כתובות חוקיות נופלות בתוך הטווח A0000h עד BFFFFh. התהליך מקבל ערך בורר אחד, או יותר, שמאפשרים לו להכנס למאגר התצוגה הפיסי. כל ערך בורר מאפשר כניסה ל-64KB של המאגר, ולכן התהליך צריך ערך בורר אחד או יותר כדי להכנס לכל המאגר שעומד לרשותו. אם גודל המאגר גדול מ-64KB, ערך הבורר הראשון יאפשר גישה ל-64KB הראשונים, ערך הבורר השני יאפשר גישה ל-64KB הבאים, וכן הלאה, כאשר ערך הבורר האחרון יאפשר גישה לשטח שנותר. התהליך יכול לטפל במאגר התצוגה הפיסי רק כאשר הוא נמצא ב-foreground session. כדי לעזור בתיאום הק/פ שלו למסך, חייב התהליך להשתמש ב-VioScrLock וב-VioScrUnLock.

VioModeUndo מבטל את השירות VioModeWait שהופעל על-ידי thread אחר בתהליך. הוא עושה זאת באחת משתי דרכים אלו: המישק גורם ל-VioModeWait להחזיר קוד שגיאה ל-thread האחר, או שהוא גורם להפסקת פעולתו של ה-thread האחר. התהליך יכול לשמור אופציה לבעלות על השירות VioModeWait, אפילו אם שירות זה בוטל (ראה פרטים ב-VioModeWait).

VioModeWait ממתין להודעה שתגרום לשחזור של אופן התצוגה, של מצב התצוגה ושל האוגרים של מתאם התצוגה. התהליך הגרפי חייב ליעד thread לביצוע הפונקציה VioModeWait, רק אם הוא כותב ישירות לאוגרים של מתאם התצוגה במטרה לשחזר את מאפייני התצוגה לאחר ש-pop-up זמני (כתוצאה משגיאה קריטית, או יישום הנמצא ברקע) הגיע לסיום. היישום הגרפי לא יצטרך לשחזר את מאגר התצוגה הפיסי, כי תת-המערכת VIO שומרת את השטח שהוקצה ל-pop-up. לאחר ש-thread בתהליך הפעיל את הפונקציה VioModeWait, הוא ממתין עד אשר שירות זה חוזר אליו. כאשר הוא חוזר, ה-thread חייב לבדוק את הקוד החוזר, ואם הוא אינו קוד שגיאה, עליו לשחזר את אופן התצוגה, את מצב התצוגה ואת האוגרים של מתאם התצוגה. ה-thread האחראי על ה-VioModeWait צריך להמנע מלבצע קריאות אחרות של המערכת או לגרום שגיאות קריטיות, מכיון שהדבר יכול לעצור את פעולת המערכת. לאחר שה-thread השלים את שחזור מאפייני התצוגה, הוא יכול להפעיל שוב את הפונקציה VioModeWait כדי שתמתין לפעם הבאה שבה הוא יצטרך לשחזר את התצוגה. ב-session ישנו רק תהליך אחד הרשאי להפעיל את VioModeWait, ומכיון שהוא עשוי לשנות את תמונת המסך, עליו להשתמש ב-VioModeUndo כדי לבטל את הפעולה.

VioSavRedrawUndo מבטל את השירות VioSavRedrawWait שהופעל על-ידי thread אחר בתהליך. הדבר נעשה באחת משתי דרכים אלו: המישק גורם ל-VioSavRedrawWait להחזיר קוד שגיאה ל-thread האחר, או שהוא גורם להפסקת פעולתו של ה-thread האחר. התהליך יכול לשמור אופציה לבעלות על השירות VioSavRedrawWait, אפילו אם השירות הנוכחי בוטל (ראה פרטים ב-VioSavRedrawWait).

VioSavRedrawWait ממתין להודעה שתגרום לשמירה, או לשחזור של מאגר התצוגה הפיסי. התהליך הגרפי חייב ליעד thread אחד לביצוע VioSavRedrawWait במטרה לשמור או לשחזר את תמונת המסך בעת מעבר בין sessions. מרגע ש-thread בתהליך הפעיל את VioSavRedrawWait, הוא ממתין עד ששירות זה יחזור אליו, ואז הוא בודק את סוג ההודעה ושומר או משחזר לפי הצורך את מאגר התצוגה הפיסי, את מצב התצוגה ו/או כל מידע אחר שדרוש ליישום כדי להשלים את תמונת המסך. ה-thread האחראי על ביצוע ההוראה צריך להמנע מלבצע קריאות מערכת אחרות או לגרום שגיאות. פעולות אלו יכולות לגרום לכך שהמערכת תעצר. בזמן הכניסה למאגר התצוגה הפיסי, ה-thread האחראי על VioSavRedrawWait אינו נועל אותו מכיון שהוא כבר קיבל את הבעלות עליו דרך מנהל ה-sessions. כאשר ה-thread השלים ביצוע שמירה/שחזור של המסך, הוא יכול לבצע שוב את VioSavRedrawWait כדי שתמתין לפעם הבאה שבה הוא יצטרך לבצע את הפעילויות האלה. ב-session ישנו רק תהליך אחד הרשאי להפעיל VioSavRedrawWait. תהליך זה עשוי לשנות את תמונת המסך ועל-כן עליו לבטל את ההוראה בעזרת VioSavRedrawUndo.

VioScrLock מבקש לקבל את הבעלות על מאגר התצוגה הפיסי. התהליך מודיע אם הוא רוצה להמתין עד אשר מאגר התצוגה הפיסי יתפנה, או שהוא רוצה שהשירות VioScrLock יחזור אליו מיידית אם המאגר אינו פנוי. התהליך

חייב לבדוק את התוצאה של ביצוע פונקציה זו, כדי לקבוע אם היישום יכול להשתמש במאגר התצוגה הפיסי. יש שתי סיבות שבגללן לא יקבל תהליך את הבעלות על המאגר:

1. ה-session שלו נמצא ברקע.
2. מאגר התצוגה הפיסי נמצא בבעלות של תהליך אחר.

ברגע שתהליך מקבל את הבעלות על מאגר התצוגה הפיסי הוא יכול לבצע אליו ק/פ. אולם, עליו לשחרר אותו מהר ככל האפשר בעזרת VioScrUnlock, כדי לאפשר למשתמש הקצה מיתוג בין ה-sessions השונים. לא ניתן למתג את ה-sessions בזמן שמאגר התצוגה הפיסי נעול. אולם, אם המשתמש מנסה לעבור ל-session אחר בזמן הנעילה, הוא ייעזר בכך שמערכת ההפעלה תגביל את הנעילה לפרק זמן מוגדר. אם פרק זמן זה עבר ובעליו של מאגר התצוגה הפיסי לא שחרר את הנעילה, מערכת ההפעלה תשעה את ביצוע התהליך (שהוא הבעלים של המאגר) ולמעשה תעביר אותו לרקע. מנקודה זו היא ממשיכה לבצע את בקשת המפעיל ולעבור session. התהליך המושעה לא יתבצע עד אשר המפעיל יחזור ל-session שבו הוא נמצא.

VioScrUnlock מבטל את הבעלות על מאגר התצוגה הפיסי.

החלפת פונקציות VIO

יישום, או תת-מערכת תצוגה אחרת, יכולים לנטרל בצורה סלקטיבית מישקים של תת-המערכת VIO במטרה להחליפם, או להרחיב את טווח פעולתם. כמובן שתת-מערכת אחרת לתצוגה יכולה להגדיל את מספר מישקי התצוגה כדי להוסיף תפקודים חדשים.

VioDeRegister מבטל את תת-המערכת לתצוגה שקודם לכן נרשמה לפעולה בתוך ה-session. רק התהליך שביצע את ה-VioRegister ב-session מסוים רשאי לבצע את VioDeRegister.

VioRegister רושם לעבודה תת-מערכת לתצוגה בתוך ה-session הנוכחי. ניתן להפעיל רק תת-מערכת אחת בו-זמנית בכל session. תת-המערכת החדשה חייבת לציין איזה מבין השירותים שמספקת תת-המערכת VIO היא עומדת להחליף. לאחר-מכן, בכל פעם שתהליך יקרא לשירות VIO שהוחלף, תקרא תת-מערכת זו לתת-המערכת החדשה והיא תבצע את הפעולה. תת-המערכת החדשה יכולה להפעיל את שירות ה-VIO המקורי על-ידי החזרה של קוד שגיאה מיוחד 1- (FFFFh) לתת-המערכת VIO.

KBD: ק/פ במקלדת

תת-המערכת KBD מתבססת על ספריית קישור דינמי, אשר המישקים שלה מתחילים בקידומת "KBD", ועל device driver למקלדת, אשר נטען אוטומטית על-ידי מערכת ההפעלה. למישקי ה-KBD יש מאפיינים רבים הדומים באופיים לשירותים המסופקים על-ידי מישקי ה-BIOS בסביבת ה-DOS. הם מאפשרים ליישום לקבל באמצעות רשומת נתוני התו את ה-scan code של צירוף הקשות מסוים ואת תו ה-ASCII המתאים ל-scan code. היישום יכול לקבל מחרוזת ASCII. רשומת נתוני התו מתוארת בתרשים 38.

קלט מהמקלדת הפיסית מוכנס למאגר בעל אורך קבוע המתוחזק על-ידי

תת-המערכת KBD. כמו ב-DOS, לאחר שהמאגר מתמלא, אין המפעיל יכול להקיש הקשות נוספות. כאשר מתרחשת גלישה (overrun) במאגר התו שאינו נכנס מתבטל. מכיון שהמקלדת הינה חלק מהקונסולה, רק יישומים הנמצאים ב-foreground session יכולים לקבל הקשות.

כפי שתזכור, בכל פעם שנפתח session, נוצר עבורו מאגר מקלדת לוגי המשותף לכל התהליכים המתבצעים באותו session. מאגר המקלדת הלוגי נקשר למקלדת הפיסית בכל פעם שה-session עובר לחזית, כלומר בכל פעם שהמפעיל עובר אליו. בכל פעם שהמפעיל עובר מ-session אחד למשנהו, תת-המערכת KBD מקשרת בין המקלדת לבין מאגר המקלדת הלוגי של ה-session החדש. מאגר המקלדת הלוגי של ה-background session מאפשר לתהליך הנמצא ב-session זה להמשיך לבצע ק/פ למקלדת. אולם, הקשות לא יוצבו במאגר המקלדת הלוגי עד אשר ה-session יועבר לחזית. יש מקרה אחד יוצא דופן בק/פ למקלדת שבו תהליך הנמצא ברקע יכול להשתמש במקלדת הפיסית: תהליך יכול להשתמש ב-foreground session זמני הנקרא pop-up לשם תקשורת עם המפעיל.

לכל session יש מאגר מקלדת שהוא ברירת מחדל, אך התהליך יכול לבחור מאגר מקלדת לוגי אחר. היישום ישתמש במקשר ברירת המחדל (0000h) של תת-המערכת KBD כדי להתחבר עם המקלדת הלוגית המוגדרת כברירת המחדל של ה-session שבו נמצא היישום. התהליך ישתמש במקלדת לוגית אחרת מזו של ברירת המחדל, אם בתוך אותו session מתבצעים תהליכים נוספים המשתמשים בהתקני הקונסולה (מסך, מקלדת ו/או עכבר). המקלדת הלוגית המשנית מאפשרת לתהליך לבודד את הק/נ שלו מזה של התהליכים האחרים. כדי שהתהליך יוכל להשתמש במקלדת הלוגית המשנית, הוא חייב להשתמש ב-KbdOpen כדי ליזום מקלדת לוגית וכדי להשיג אליה מקשר. התהליך משתמש ב-KbdGetFocus כדי לקשר באופן זמני את המקלדת הלוגית למקלדת הפיסית לצורך קבלת קלט. KbdFreeFocus מנתק את המקלדת הלוגית מהמקלדת הפיסית ומאפשר לתהליכים אחרים להשיג את ה-Focus. KbdClose מנתק את התהליך מהמקלדת הלוגית.

מקשרי ה-KBD למקלדת אינם ניתנים לירושה כמו המקשרים שהוקצאו על-ידי מערכת הקבצים.

מישקי ה-KBD המתוארים מטה חולקו למספר קבוצות כלליות והן:

- ק/פ של תווים.
- בקרה על ההתקן.
- ניהול המקלדת הלוגית.
- החלפת פונקציות.

KBD ק/פ של תווים

היישום מבצע קלט ממקלדת באמצעות מקשר ה-KBD למקלדת לוגית המוגדרת כברירת מחדל, או באמצעות מקשר KBD למקלדת לוגית אחרת. התהליך יכול לקרוא רשומת תו, או מחרוזת של תווים, ממאגר הקלט של המקלדת הפועל לפי שיטת FIFO.

0	קוד ASCII של התו	Scan Code	סוג התו
3	מצב NLS Shift	מצב מקש ה-Shift	
6	חותמת הזמן של התו		

- קוד ה-ASCII של התו מתורגם מה-scan code.

- ה-scan code מתקבל מהמקש שנלחץ.

- סוג התו מגדיר את הדברים הבאים:

  - תו סיום
  - תו ביניים (נתמך על-ידי NLS)
  - אין תו, מקש shift בלבד.

- מצב NLS Shift נשמר (00h).

- מצב מקש ה-Shift מודיע על מצב המקשים הבאים:

SysReq	Right Alt	Insert
CapsLock	Left Alt	Right Shift
NumLock	Right Ctrl	Left Shift
Scroll Lock	Left Ctrl	

- חותמת הזמן של המקש מבוטאת באלפיות שניה מאז ה-IPL האחרון.

תרשים 38. רשומת נתוני תו

KbdCharIn קורא רשומה של נתוני תו מן המקלדת הלוגית (לפי ברירת המחדל, או אחרת), המזוהה על-ידי מקשר ה-KBD שלה אל תוך מאגר שצוין על-ידי התהליך. התהליך חייב גם להודיע אם הוא רוצה להמתין עד אשר התו יתקבל, או שהוא רוצה שהשירות KbdCharIn יחזור מיד, אם אין אף לא תו אחד. רשומת נתוני תו תחזור רק אם המקלדת הלוגית קשורה באותו רגע למקלדת הפיסית. סביר להניח שהמקלדת הלוגית, המוגדרת כברירת המחדל ב-session, תהיה קשורה, אלא אם נעשה שימוש במקלדת לוגית אחרת.

KbdFlushBuffer מנקה את מאגר המקלדת הלוגית (ברירת המחדל או אחר) המזוהה על-ידי מקשר ה-KBD. הניקוי יתבצע כאשר המקלדת הלוגית קשורה

באותו רגע למקלדת הפיסית, וסביר להניח שהמקלדת הלוגית המוגדרת כברירת המחדל תהיה ממוקדת, אלא אם נעשה שימוש במקלדת לוגית אחרת.

KbdPeek מחזיר רשומת נתוני תו מהמקלדת הלוגית המזוהה על-ידי מקשר ה-KBD שלה. רשומה זו אינה מוצאת מהמאגר. רשומת נתוני המקלדת תוחזר אך ורק אם המקלדת הלוגית קשורה באותו רגע למקלדת הפיסית, וסביר להניח שהמקלדת הלוגית המוגדרת כברירת המחדל תהיה ממוקדת, אלא אם נעשה שימוש במקלדת לוגית אחרת.

KbdStringIn מחזיר מספר מוגדר מראש של תווי ASCII (רק את הקודים עצמם) ממקלדת לוגית המזוהה בעזרת מקשר ה-KBD שלה. מספר התווים (בתים) המקסימלי הוא 255. התהליך חייב להודיע אם הוא רוצה להמתין או לא:

WAIT במצב של קלט בינארי, התהליך ממתין עד אשר הוא מקבל את מספר התווים שביקש. במצב של קלט ב-ASCII, התהליך ממתין עד אשר יוקש תו "שורה חדשה" (CarriageReturn).

NO WAIT במצב של קלט בינארי, התהליך מקבל מספר תווים רב ככל האפשר, עד למספר המצוין בבקשה. אם לא ניתן אף תו, השירות KbdStringIn חוזר מיד. במצב של קלט ב-ASCII, אין תמיכה לאופציה NO WAIT.

תווי המקלדת יוחזרו אך ורק אם המקלדת הלוגית קשורה באותו רגע למקלדת הפיסית. סביר להניח שהמקלדת הלוגית, המוגדרת כברירת המחדל ב-session, תהיה ממוקדת, אלא אם נעשה שימוש במקלדת לוגית אחרת.

KbdXlate מתרגם את ה-scan code ואת מצבי מקש ה-Shift שנמצאו ברשומת נתוני התו לקוד ASCII. התרגום מתבצע לפי דף הקוד (code page) המתאים למקלדת הלוגית המזוהה בעזרת מקשר ה-KBD שלה. התו המתורגם מוחזר במסגרת רשומת נתונים מתורגמים, המוצבת במאגר שצוין על-ידי היישום. בגמר התרגום יופעלו אותות סימון (flags). אם יוקשו צירופי מקשים מסוימים, יהיה צורך לקרוא למישק זה מספר פעמים.

KBD בקרה על ההתקן

יישום יכול להשתמש במספר מישקים שיאפשרו לו לקבל ולקבוע פרטים מסוימים על המקלדת ועל הדרך שבה יטופלו ההקשות.

KbdGetCP מחזיר את מספר הזיהוי של דף הקוד המשמש לתרגום ה-scan codes לתווי ASCII עבור המקלדת הלוגית המזוהה בעזרת מקשר KBD.

KbdGetStatus מחזיר מאפיינים של המקלדת ביחס למקלדת לוגית המזוהה בעזרת מקשר KBD. המידע על ההתקן מוצב במאגר שצוין על-ידי התהליך, והוא כולל את הפרטים הבאים:

- מצב הקלט.
  מצב הקלט יכול להיות ASCII או בינארי. במצב ASCII, תווים המתקבלים מהמקלדת מבוקרים בעזרת תווי בקרה מיוחדים (כמו linefeed, למשל), הגורמים לביצוע פעולה מסוימת בדומה לפקודה. במצב בינארי, אין בקרה על התווים.

- סמן תו הביניים.
אפשר להפעיל את סמן תו הביניים (Interim Character Flag) כדי להודיע שהיישום רוצה לקבל כל תו ביניים השייך לתו מרובה סימנים, במטרה לבנות את התו הסופי. סמן זה הוא בעל משמעות רבה לשפות, אשר דרוש בהן לבצע מספר הקשות כדי לבנות תו אחד.
- מצב מקש Shift.
מצב מקש Shift מציין אם השתמשו באחד ממקשי ה-Shift. לדוגמה, לחיצה על מקש CapsLock תקבע את המצב שלו ל-ON, ולחיצה נוספת על מקש זה תקבע את המצב ל-OFF. מקשי ה-Shift כוללים את המקשים הבאים:

SysReq	Right Alt	Insert
CapsLock	Left Alt	Right Shift
NumLock	Right Ctrl	Left Shift
Scroll Lock	Left Ctrl	

- מצב ה-Echo.
מצב ה-Echo מציין אם תווי המקלדת יוצגו באמצעי הפלט הסטנדרטי.
- הגדרת תו השורה החדשה.
הגדרת תו השורה החדשה מצביעה על התו, ASCII או ASCII מורחב, שמציין מעבר לשורה חדשה, או במילים אחרות את תו "שורה חדשה" - CarriageReturn.

KbdSetCP קובע את דף הקוד המזוהה בעזרת מספר הזיהוי שלו. דף זה משמש לתרגום ה-scan codes לתווי ASCII, אשר נעשה עבור מקלדת לוגית המזוהה בעזרת מקשר KBD. מאגר המקלדת הלוגי מרוקן את התווים שתורגמו בעזרת דף הקוד הקודם.

KbdSetCustXt קובע את דף הקוד המשמש לתרגום טבלה שסופקה על-ידי התהליך לצורך עבודת המקלדת הלוגית המזוהה על-ידי מקשר KBD. התהליך חייב לשמור את הטבלה המתורגמת בזיכרון הנמצא בבעלותו. מאגר המקלדת הלוגי מרוקן את התווים שתורגמו בעזרת דף הקוד הקודם.

KbdSetStatus קובע את מאפייני המקלדת הלוגית המזוהה בעזרת מקשר KBD. המידע על ההתקן מועבר במאגר שצוין על-ידי התהליך והוא כולל את מצב הקלט, סמן תו הביניים, מצב מקש ה-Shift, מצב ה-Echo, ואת הגדרת תו השורה החדשה, כפי שתוארו ב-KbdGetStatus.

ניהול של המקלדת הלוגית KBD

יישום יכול לטפל במקלדת לוגית השונה מברירת המחדל של ה-session.

KbdClose סוגר את המקלדת הלוגית (ברירת המחדל או אחרת) המזוהה על-ידי מקשר KBD. המקלדת הלוגית המוגדרת כברירת המחדל, ניתנת לסגירה בידי התהליך, אך היא תמשיך להתקיים לגבי ה-session. מקלדת לוגית משנית תנותק, לפי הצורך, מהמקלדת הפיסית ותרוקן לפני שפעולתה תופסק.

KbdFreeFocus מנתק את המקלדת הלוגית, המזוהה על-ידי המקשר שלה, מהמקלדת הפיסית. אם אין אף בקשה ממתינה לביצוע מיקוד (focus), המיקוד חוזר למקלדת הלוגית שהיא ברירת המחדל.

KbdGetFocus מחבר את המקלדת הלוגית המזוהה על-ידי מקשר KBD אל המקלדת הפיסית. התהליך חייב לציין אם הוא רוצה להמתין שהמקלדת הפיסית תתפנה, או שהוא רוצה שהשירות KbdGetFocus יחזור אליו מיד, אם המיקוד של המקלדת הפיסית נמצא כבר בבעלות של מקלדת לוגית אחרת.

KbdOpen מחזיר מקשר KBD למקלדת הלוגית המשנית. המקלדת הלוגית מאותחלת לדף הקוד המוגדר כברירת המחדל.

החלפת פונקציות KBD

יישום או תת-מערכת אחרת למקלדת יכולים לנטרל את מישקי ה-KBD במטרה להרחיב או להחליף את השירותים המסופקים על-ידי תת-המערכת KBD. כמובן, שכמו בכל תת-מערכת, תת-מערכת נוספת למקלדת יכולה גם להוסיף מישקים נוספים לאלה הקיימים.

KbdDeRegiser מבטל את הרישום של תת-המערכת לפעילות ב-session. רק התהליך שביצע את ה-KbdRegister יוכל לבצע את ה-KbdDeRegister.

KbdRegister רושם לעבודה תת-מערכת למקלדת בתוך ה-session הנוכחי. בכל session ניתן להפעיל רק תת-מערכת אחת, אשר חייבת להודיע איזה מבין שירותי הק/פ של תת-המערכת KBD היא עומדת לנטרל בזמן שהיא פועלת. יש לציין, שתת-המערכת החדשה תשפיע אך ורק על התהליכים הנמצאים ב-session הזהה לשלה. מרגע זה, בכל פעם שתהליך יפעיל שירות של KBD שנוטרל, תקרא תת-המערכת KBD לתת-המערכת החדשה. לגבי שירות ה-KBD שנוטרל, תת-המערכת החדשה יכולה לגרום לביצוע שירות זה על-ידי החזרת קוד שגיאה מיוחד בעל ערך של 1- (FFFFh). תת-המערכת KBD תקרא לשירות ה-KBD המקורי.

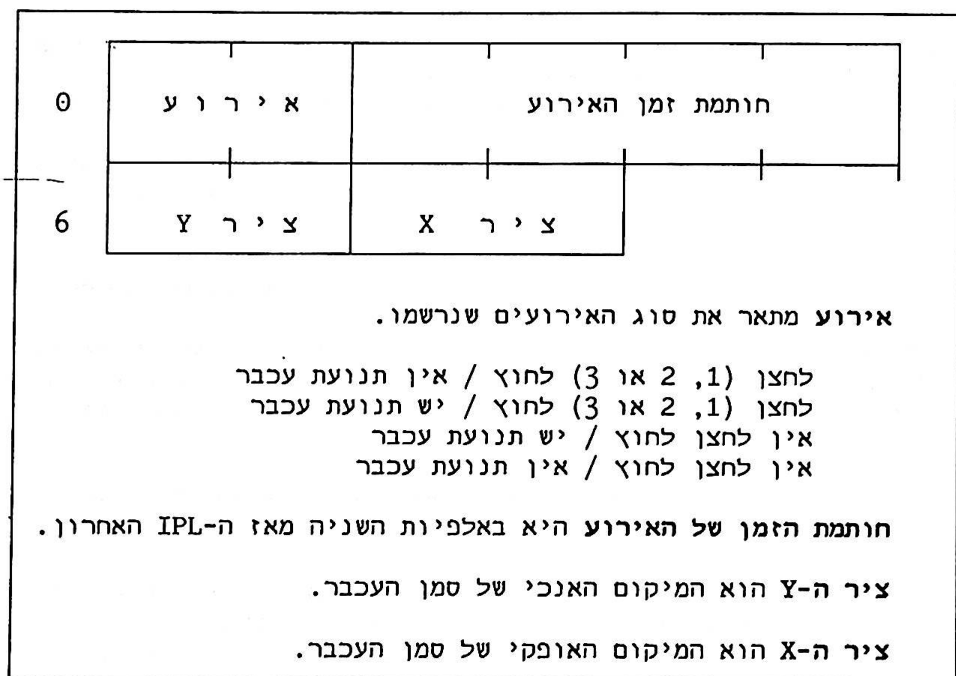
KbdSynch מאפשר לתת-המערכת החדשה שנרשמה לעבודה לסנכרן את הגישה שלה למקלדת הפיסית. פונקציה זו עוזרת לתת-מערכות המטפלות במקלדת לתאם את הפעולות שלהן עם ה-device driver של המקלדת.

MOU: ק/פ באמצעות העכבר

תת-מערכת MOU מתבססת על ספריית קישור דינמית, אשר מישקיה מתחילים בקידומת "MOU", ועל שני device drivers: של העכבר ושל מחוון הציור (pointer draw). יש צורך להטעין את ה-device drivers באמצעות הפקודה =DEVICE ב-CONFIG.SYS. תת-מערכת MOU מאפשר ליישום לשלוט בפעילויות הבאות:

- בדרך שבה יטופל הסמן של העכבר.
- בסוג של אירועי עכבר, אשר יכללו כנתונים ויועברו למאגר הקלט (buffer).
- בפעולות אחרות הקשורות בניהול ההתקן.

רשומת נתוני העכבר מתוארת בתרשים 39.



תרשים 39. רשומת נתוני העכבר

קלט מהעכבר הפיסי מוצב במאגר המתוחזק על-ידי תת-מערכת MOU. נתוני העכבר מגיעים בצורה של אירוע, המתבסס על לחיצה של לחצן אחד, או יותר, ומשולב עם התנועה של העכבר. סמן העכבר (נקרא גם מחוון העכבר) במסך עוקב אחר התנועה של העכבר הפיסי, אשר יכולה להיות מדווחת בקואורדינטות, או ב-mickeys. המונח mickey הוא יחידה יחסית של תנועת העכבר. התהליך יכול לבחור את סוג היחידות שישמשו במדידת תנועת העכבר. השימוש בקואורדינטות הינו ברירת המחדל של המערכת להצבעה על תנועת העכבר. קואורדינטות ניתנות בתווים, או במרחק יחסי בפיקסלים, בהתאם למצב המסך. ערכיהם יחסיים לפינה השמאלית העליונה של המסך, שהיא נקודת ה-0 לגבי צירי ה-X וה-Y. אם הוחלט להשתמש ב-mickeys, ערכי ה-X וה-Y יהיו יחסיים למיקום הנוכחי של העכבר. ערכי mickey שליליים מצביעים על תנועה למעלה או שמאלה במסך. משתמש הקצה מזיז את העכבר וגורם למאגר להתמלא עם הנתונים הנובעים מתנועה זו. אם המשתמש ימשיך להזיז את העכבר, התנועות שלו לא יאבדו, אלא ייכתבו על הנתונים הקודמים במאגר. במלים אחרות, למרות שהמאגר הינו בעל גודל קבוע, הוא בעל אופי "מעגלי" ושומר על האירועים האחרונים של העכבר. מכיון שהעכבר הינו חלק מהקונסולה, רק יישומים הנמצאים ה-foreground session יוכלו לקבל ממנו נתונים.

כפי שתזכור, בכל פעם שנפתח session נוצר גם מאגר עכבר לוגי, אשר משותף לכל התהליכים המתבצעים ב-session. לכל session יש מאגר עכבר לוגי משלו. מאגר העכבר הלוגי נקשר לעכבר הפיסי בכל פעם שה-session עובר להיות foreground session. בכל פעם שהמפעיל עובר מ-session אחד למשנהו, תת-מערכת ה-MOU מחברת את העכבר הפיסי למאגר העכבר הלוגי של

ה-session החדש. מאגר העכבר הלוגי השייך ל-background session מאפשר לכל תהליך השייך אליו להמשיך בק/פ לעכבר; אולם, נתוני העכבר לא יוצבו במאגר העכבר הלוגי של ה-background session עד אשר ה-session יעבור לחזית. ישנו רק מקרה יוצא דופן אחד שבו תהליך הנמצא ברקע יוכל להשתמש בעכבר הפיסי: תהליך ברקע יכול להשתמש ב-foreground session מיוחד הנקרא pop-up, כדי להתקשר עם המפעיל.

הפרוטוקול המשמש לביצוע הק/פ לעכבר דומה לזה המשמש התקני תו אחרים. התהליך פותח את העכבר בעזרת MouOpen, אשר מאתחל את העכבר הלוגי. מרגע זה הוא מבצע את כל פעולות הק/פ הקשורות בעכבר בעזרת מקשר MOU שמוחזר על-ידי MouOpen. לבסוף, התהליך מפעיל את MouClose כדי להפסיק את הקשר לעכבר הלוגי. כדי לטפל באירועי העכבר, התהליך חייב לעקוב אחר סוגי האירועים שדווחו. לדוגמה, נניח שהתהליך השתמש ב-MouSetEventMask כדי להודיע שהוא רוצה שרק אירועים הקשורים ללחצן 1 ידווחו. כדי להמחיש דוגמה זו, הצגנו בתרשים 40 אירועים אשר דווחו במאגר הקלט של העכבר על-פי פעולות שננקטו על-ידי המפעיל.

מישקי ה-MOU חולקו למספר קטגוריות כלליות:

- ק/פ של נתונים.
- בקרת הסמן.
- בקרת ההתקן.
- החלפת פונקציות.

MOU ק/פ של נתונים

יישום יוכל לקבל קלט מעכבר לאחר שישיג מקשר MOU להתקן. הקלט עצמו הינו קריאה של רשומות נתונים של העכבר ממאגר הקלט הפועל לפי שיטת FIFO.

MouClose סוגר את העכבר המזוהה על-ידי מקשר ה-MOU שלו.

MouFlushQue מנקה את המאגר המכיל את אירועי העכבר המזוהה בעזרת מקשר MOU.

MouGetNumQueEl מחזיר את מספר רשומות נתוני העכבר הנמצאים כרגע בתוך מאגר הקלט של העכבר. הוא גם מחזיר את מספר הרשומות המקסימלי היכול להיות במאגר הקלט של העכבר המזוהה בעזרת מקשר MOU.

MouOpen פותח את העכבר לתהליך ומחזיר מקשר MOU להתקן.

MouReadEventQue קורא רשומת נתוני עכבר ממאגר הקלט של העכבר המזוהה בעזרת מקשר MOU. התהליך חייב להודיע אם הוא רוצה להמתין עד אשר הוא יקבל רשומה, או שהוא רוצה שהשירות MouReadEventQue יחזור מיד אם אין כל רשומה שהיא. ראה תרשים 39 המתאר את מבנה רשומת נתוני העכבר.

פעולת המפעיל	סוג האירוע שדווח
לחצן 1 נלחץ.	לחצן 1 לחוץ. אין תנועה.
לחצן 1 עדיין לחוץ. העכבר בתנועה.	לחצן 1 לחוץ. יש תנועה.
לחצן 1 משוחרר. העכבר נעצר.	אף לא לחצן אחד לחוץ. אין תנועה.
לחצן 2 נלחץ.	אין אירוע.
לחצן 2 עדיין לחוץ. העכבר בתנועה.	אין אירוע.
לחצן 2 עדיין לחוץ. העכבר עדיין בתנועה. לחצן 1 נלחץ.	לחצן 1 לחוץ. לחצן 2 לחוץ. יש תנועה.
לחצן 1 שוחרר. לחצן 2 שוחרר. העכבר עדיין בתנועה.	אף לא לחצן אחד לחוץ. יש תנועה.
העכבר נעצר.	אין אירוע. (אין שינוי הקשור ללחצן 1)

תרשים 40. דוגמה לאירועי עכבר המבוססים על פעולות המפעיל

MOU בקרה על הסמן

יישום יכול לשלוט בצורה ובמיקום של סמן העכבר, כמו גם באיזורים שבהם ניתן להציג את סמן העכבר.

MouDrawPtr מאפשר לסמן של העכבר המזוהה בעזרת מקשר MOU לנוע בשטח שהיה סגור בפניו קודם. פעולה זו גורמת לשרטוט סמן העכבר אם הוא נמצא באיזור שנסגר בפניו קודם. MouRemovePtr קובעת את האיזור שייסגר בפני סמן העכבר.

MouGetPtrPos מחזיר את המיקום האנכי והאופקי של הסמן של העכבר המזוהה בעזרת מקשר MOU.

MouGetPtrShape מחזיר העתק של צורת הסמן המשמש את העכבר המזוהה בעזרת מקשר MOU. ברירת המחדל במקרה של מסך במצב טקסט היא ריבוע, שמאפשר לתווים שכתובים כבר במסך להמשיך להראות. ברירת המחדל במקרה של מסך במצב גרפי היא חץ. התהליך יכול לשנות את הצורה של סמן העכבר בעזרת MouSetPtrShape.

MouRemovePtr קובע את שטח המסך שיעמוד לרשות הסמן של העכבר. התהליך מגדיר ריבוע שבו הסמן לא יוכל להופיע, ואם הוא נמצא בתוך האיזור המוגבל, הוא מוצא משם. MouDrawPtr משחרר את ההגבלות על התנועה של הסמן.

MouSetPtrPos קובע את המיקום האנכי והאופקי של סמן העכבר, המתאים לעכבר המזוהה בעזרת מקשר MOU.

MouSetPtrShape קובע את הצורה שבה יופיע סמן העכבר המתאים לעכבר המזוהה בעזרת מקשר MOU. התהליך חייב לציין את הפרטים הבאים:

- סיביות המיסוך XOR ו-AND לצורך קביעת הצורה של הסמן.
- הרוחב והגובה של הסמן המוצע.
- נקודת ההתחלה של הסמן.

MOU בקרה על ההתקן

לרשות היישום עומדים מספר מישקים המאפשרים לקבל ולקבוע פרטי מידע על העכבר ועל מאגר הקלט שלו. היישום יכול לשלוט בסוג האירועים שידווחו, ומי יציג את סמן העכבר.

MouGetDevStatus מחזיר את מצב העכבר המזוהה בעזרת מקשר MOU. המידע הזה כולל את הפרטים הבאים:

- האם המיקום של העכבר מצוין בקואורדינטות או ב-mickeys.
- האם התהליך אחראי לתחזוקה של הסמן במסך.
- האם ניתן לשרטט את העכבר במצב המסך הנוכחי.
- האם מאגר האירועים של העכבר מתרוקן כרגע.
- האם מתבצעת כרגע קריאה ממאגר האירועים של העכבר.
- האם מאגר האירועים של העכבר עסוק כרגע עם ק/פ.

MouGetEventMask מחזיר את סוגי האירועים המדווחים באמצעות רשומות נתוני עכבר השייכות לעכבר המזוהה בעזרת מקשר MOU. הזיהוי של הלחצנים נעשה באמצעות מספר לוגי, אשר השמאלי בהם הוא 1. סוגי האירועים הם כדלהלן:

- לחצן 1 לחוץ/משוחרר.
- לחצן 1 לחוץ/משוחרר וקיימת תנועה של העכבר.
- לחצן 2 לחוץ/משוחרר.
- לחצן 2 לחוץ/משוחרר וקיימת תנועה של העכבר.
- לחצן 3 לחוץ/משוחרר.
- לחצן 3 לחוץ/משוחרר וקיימת תנועה של העכבר.
- קיימת תנועה של העכבר ללא פעילות בלחצנים.

MouGetHotKey מציין את מקשי העכבר המקבילים בתפקידם לצירופים של מקשים במקלדת הידועים בשם "מקשים חמים". מקשים אלה מוגדרים לכל המערכת, ועל כן הם זהים בכל ה-sessions. הם מאפשרים למפעיל לפנות לבורר התכניות של OS/2, אשר מראה רשימה של יישומים שניתן להתחיל אותם ורשימה של ה-sessions הפעילים כרגע. הוא גם מאפשר לעבור session. המקש החם של העכבר יכול להיות מורכב מלחצן אחד, או יותר, שחייבים להלחץ בו-זמנית.

MouGetNumButtons מחזיר את מספר הלחצנים בעכבר.

MouGetNumMickeys מחזיר את מספר ה-mickeys הנכנסים בסנטימטר אחד לגבי העכבר המזוהה בעזרת מקשר MOU.

MouGetScaleFact מחזיר את היחס שבין התנועה האנכית והאופקית של העכבר, לבין תנועת הסמן של העכבר על פני המסך. יחס זה מבטא את מספר ה-mickeys הדרושים להזיז את סמן העכבר לאורך 8 פיקסלים. ברירת המחדל לציר האנכי היא 8 יחידות mickey עבור 8 פיקסלים. ברירת המחדל לציר האופקי היא 16 יחידות mickey עבור 8 פיקסלים.

MouSetDevStatus קובע פרמטרים מסוימים לעכבר המזוהה בעזרת מקשר MOU. הפרמטרים שניתן לקבוע אותם הם:

- האם המיקום של העכבר יצוין בקואורדינטות, או ב-mickeys.
- האם התהליך יהיה אחראי לתחזוקת הסמן במסך.

MouSetEventMask קובע את סוגי האירועים שידווחו באמצעות רשומות נתוני עכבר השייכות לעכבר המזוהה בעזרת מקשר MOU. הזיהוי של הלחצנים נעשה באמצעות מספר לוגי, אשר השמאלי בהם הוא 1. סוגי האירועים שניתן לדווח עליהם הם:

- לחצן 1 לחוץ/משוחרר.
- לחצן 1 לחוץ/משוחרר וקיימת תנועה של העכבר.
- לחצן 2 לחוץ/משוחרר.
- לחצן 2 לחוץ/משוחרר וקיימת תנועה של העכבר.
- לחצן 3 לחוץ/משוחרר.
- לחצן 3 לחוץ/משוחרר וקיימת תנועה של העכבר.
- קיימת תנועה של העכבר ללא פעילות בלחצנים.

MouSetScaleFact קובע את היחס שבין תנועת העכבר במציאות לבין תנועת הסמן במסך. יחס זה מבטא את מספר ה-mickeys הדרושים להזיז את סמן העכבר ב-8 פיקסלים.

החלפת פונקציות MOU

יישום או תת-מערכת אחרת לעכבר יכולים לנטרל את מישקי ה-MOU, במטרה להרחיב או להחליף את השירותים המסופקים על-ידי תת-המערכת MOU. כמובן שכמו בכל תת-מערכת, תת-מערכת נוספת למקלדת יכולה להוסיף מישקים לאלה הקיימים.

MouDeRegiser מבטל את הרישום של תת-המערכת שנרשמה קודם לפעולה ב-session. רק התהליך שביצע את ה-MouRegister ב-session יוכל לבצע את ה-MouDeRegister.

MouRegister רושם לפעולה תת-מערכת לעכבר בתוך ה-session הנוכחי. בכל session ניתן להפעיל רק תת-מערכת אחת, אשר חייבת להודיע איזה מבין שירותי הק/פ של תת-המערכת MOU היא עומדת לנטרל בזמן שהיא פועלת. יש לציין שתת-המערכת החדשה תשפיע אך ורק על התהליכים הנמצאים ב-session הזהה לשלה. מרגע זה, בכל פעם שתהליך יפעיל שירות של MOU שנוטרל, תקרא תת-המערכת MOU לתת-המערכת החדשה. לגבי שירות ה-MOU שנוטרל, תת-המערכת

החדשה יכולה לגרום לביצוע שלו על-ידי החזרת קוד שגיאה מיוחד בעל ערך של 1- (FFFFh). תת-המערכת MOU תקרא אז לשירות ה-MOU המקורי.

MouSynch מאפשר לתת-המערכת החדשה הרשומה לפעולה לסנכרן את הגישה שלה לעכבר הפיסי.

אפשרויות לבקרת ק/פ (IOCtl)

יישום יכול לפקח על ההתקן ועל הפעולות שלו באמצעות מישק בקרת הק/פ. מישק זה ידוע גם בשם מישק I/O Control - IOCtl, זאת בנוסף לאפשרויות הקיימות באמצעות מישקי מערכת הקבצים ומישקים של תת-מערכות אחרות המטפלים בק/פ. מישקי בקרת הק/פ משמשים לבקרה על הפרמטרים של ה-device driver הניתנים לשינוי. מישק בקרת הק/פ קשור קשר הדוק לסוג ההתקן, ולכן הפקודות שלו מחולקות לקטגוריות לפי סוג ההתקן, ובתוך כל קטגוריה יש חלוקת משנה לפי פונקציות. בתרשים 41 תמצא רשימה של הקטגוריות שמוגדרות כרגע על-ידי OS/2.

יישום ב-OS/2 מגיש בקשה ל-IOCtl באמצעות הפונקציה DosDevIOCtl. תנאי מוקדם לבקשת IOCtl הוא קבלת מקשר להתקן. זאת אומרת, שהיישום חייב קודם לפתוח את ההתקן כדי לקבל מקשר של מערכת הקבצים להתקן. במקרה של התקני תו, היישום משתמש בשמו של התקן התו עם שיגרת הביצוע OPEN של מערכת הקבצים. כדי לפתוח התקנים המטפלים בבלוקים, התהליך משתמש בשיגרת הביצוע OPEN של מערכת הקבצים עם אות הכונן המשמשת כשם ההתקן. לאחר שהיישום משיג מקשר להתקן, הוא יכול להשתמש בפקודת IOCtl לאותו התקן שהוא השיג את המקשר שלו. מערכת ההפעלה מקבלת את בקשת ה-IOCtl ומעבירה אותה ל-device driver של ההתקן שמדובר בו.

Hex	קטגוריה לפי התקן
01	בקרה על התקן סידרתי
03	בקרה על מחוון הציור
04	בקרה על המקלדת
05	בקרה על המדפסת
07	בקרה על העכבר
08	בקרה על דיסק/דיסקט לוגי
09	בקרה על הדיסק הפיסי
0A	בקרה על התקן הפיקוח
0B	בקרה כללית

תרשים 41. קטגוריות ה-IOCtl

יישום ב-OS/2 אינו חייב לשלוח פקודות IOCtl להתקנים שהוא משתמש בהם. מבין הקטגוריות השונות שכרגע מוגדרות, היישום צריך לשקול שימוש ב-IOCtl של ההתקן הסידרתי או בזה של המדפסת, אם הוא משתמש בהתקנים אלה. יישום ה-OS/2 יוכל לוותר על ה-IOCtl של המקלדת, העכבר ומחוון הציור ולשלוט בהתקנים אלה דרך תת-המערכות הייעודיות של כל אחד מהם. תת-המערכות KBD ו-MOU צריכות לבחון את השימוש ב-IOCtl עבור התקנים אלה. השיגרה של המערכת המבצעת פירמוט, או זו המבצעת מחיצות במצעים המגנטיים, חייבת להשתמש ב-IOCtl של הדיסק/דיסקט הלוגי ובאלה של הדיסק הקשיח בהתאמה. ה-IOCtl הכללי וזה של התקן הפיקוח משמשים בד"כ את מערכת ההפעלה, או מרכיבים אחרים של המערכת.

קטגוריות IOCtl נוספות וחלוקת המשנה בתוכן ניתנות להגדרה על-ידי יישום או תת-מערכת שמספקים device driver. קטגוריית ה-IOCtl הינה שדה בגודל בית, שבו הסיבית החשובה היא הסיבית הגבוהה ביותר (סיבית 7). כאשר היא מופעלת, היא מצביעה על כך שהיישום הגדיר את הקטגוריה. אם היא ריקה, הדבר מצביע על כך שמערכת ההפעלה הגדירה את הקטגוריה. בתרשים 42 תמצא את הגדרת הסיביות בקוד של הקטגוריה. פונקציות המשנה של ה-IOCtl הן שדה בגודל של בית, שבו שלוש הסיביות הגבוהות ביותר הן המשמעותיות ביותר. סיבית 7 מורה איך לטפל בבקשה, אם ה-device driver אינו תומך בפונקציית המשנה המסוימת. סיבית 6 מורה אם פונקציית המשנה נשלחת ל-device driver לצורך עיבוד. סיבית 5 מציינת אופציה בלבד ומטרתה לארגן את הקודים בפונקציית המשנה. בתרשים 42 תמצא את ההגדרה של המספר המייצג את פונקציית המשנה.

פונקציית משנה								קוד קטגוריה							
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
*	*	*	*	*	G	D	I	*	*	*	*	*	*	*	U

U = 1 מוגדר על-ידי המשתמש
0 מוגדר על-ידי המערכת

I = 1 התעלם מהפקודה, אם אינה נתמכת על-ידי ההתקן
0 החזר שגיאה, אם אינה נתמכת על-ידי ההתקן

D = 1 שלח ל-device driver
0 שלח למערכת ההפעלה

G = 1 השג נתונים מההתקן (בחירה)
0 שלח נתונים להונקן (בחירה)

* = 1 ניתן להגדרה
0

תרשים 42. ההגדרה של הקטגוריה ופונקציית המשנה ב-IOCtl

לסיכום, כדי להגדיר קטגוריה חדשה, יישום או תת-מערכת, צריכים להציב בסיבית הגבוהה ביותר 1, ולהשתמש בסיביות הנותרות, לפי הצורך, כדי ליצור מספר קטגוריה. כדי להגדיר פונקציית משנה, היישום או תת-מערכת צריכים לקבוע מה יהיו הערכים בשתי הסיביות הגבוהות ביותר, ולהשתמש בשאר שש הסיביות להגדרת מספר הפונקציה.

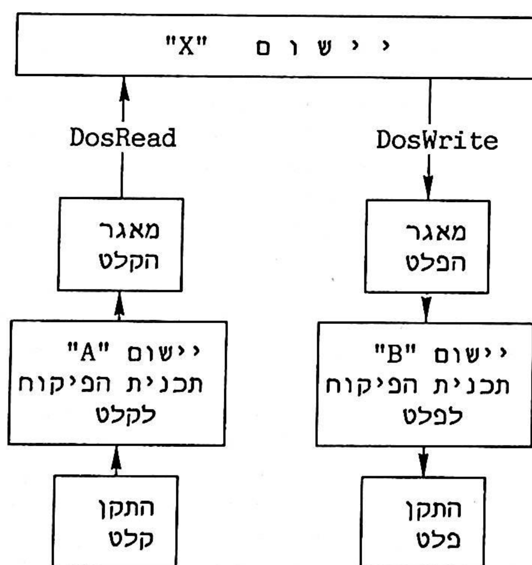
תכניות פיקוח להתקני תו

תכנית פיקוח (monitor) להתקן תו היא מנגנון המאפשר ליישום OS/2, או לתת-מערכת, "ליירט" את זרם הנתונים העובר דרך התקן (ראה תרשים 43). היישום יכול לשנות, או לצרוך חלק מהנתונים, או את כולם, באחד משני המקרים הבאים:

- לפני שהנתונים מוכנסים למאגר ההתקן, אשר קוראים ממנו נתונים.
- לפני שהנתונים שנכתבו על-ידי יישומים אחרים נשלחים להתקן, אשר כותבים אליו נתונים.

יישום DOS משתמש בשיטה אחרת כדי להשיג את אותה פעולה: יישום DOS יכול להשתלט על פסק של ההתקן, או על פסק של ה-BIOS, כדי שהוא יוכל לדעת מתי הנתונים עוברים דרך ההתקן.

יישום "המיירט" את זרם הנתונים של ההתקן נותן למעשה שירות ליישומים אחרים. תכנית פיקוח להתקני תו היא כלי, שהיישום יכול להשתמש בו לביצוע שירות שיהיה שקוף ליישומים אחרים. אין צורך לשנות יישומים אחרים, כדי שיוכלו להשתמש בשירותים אלה; הם ממשיכים לבצע את פעולות הק/פ שלהם כרגיל.



תרשים 43. התנועה של נתוני ההתקן באמצעות תכניות הפיקוח

תכנית הפיקוח להתקני תו תלויה ב-device driver לביצוע פעולותיה: אין זו רק הגדרה של אופי זרם הנתונים אלא הוא גם חייב להשתתף באופן פעיל בהכוונת הנתונים לתכנית הפיקוח. הגדרת של זרם הנתונים חשובה ליחסים בין ההתקן למערכת (וכתוצאה מכך, גם עם תכנית הפיקוח), כי הוא מגדיר איך יישומים יבצעו ק/פ להתקן. לדוגמה, זרם הנתונים יכול להיות מטופל ביחס ל-session, כפי שהדבר נעשה עם המקלדת והעכבר, או ביחס לכלל המערכת כפי שהדבר נעשה עם המדפסת. זאת אומרת, שהיקף ההשפעה של תכנית הפיקוח על הק/פ של יישומים אחרים הוא בהתאם להיקף של זרם הנתונים לאותו התקן. לכן, תכנית פיקוח על זרם נתונים המוגדר ביחס ל-session מסוים תשפיע רק על אותם יישומים הנמצאים באותו session. תכנית פיקוח על זרם נתונים המוגדר לכל המערכת תשפיע על כל היישומים בכל ה-sessions. הכוונת זרם הנתונים שמבוצעת על-ידי ה-device driver חשובה, מאחר ואין ביכולתה של תכנית הפיקוח לשלוט בזרם הנתונים. זאת אומרת, שאפשר להשתמש בתכניות פיקוח רק בהתקנים שה-device driver שלהם תומך בפיקוח.

ב-OS/2 יש שלושה device drivers שמאפשרים פיקוח על זרם הנתונים שלהם: המקלדת, העכבר והמדפסת. ה-device drivers של המקלדת והעכבר מגדירים זרם נתונים לכל session. ה-device driver של המדפסת מגדיר זרם נתונים לכלל המערכת עבור התקן אחד. לדוגמה, היישום שמפקח על נתוני המקלדת יכול למפות (map) את צירופי המקשים, או לקבוע זיקה בין צירוף מקשים אחד לשני. זאת אומרת, שהקשה אחת או יותר תוכל לשמש כקיצור לצירוף של מספר הקשות. מאחר וקיים רק זרם נתוני מקלדת אחד בכל session, המפעיל יוכל להתאים קיצורים שונים ליישומים הנמצאים ב-sessions נפרדים. באשר למדפסת, היישום יוכל להשתמש בתכנית הפיקוח, כדי "ליירט" את המידע להדפסה ולכוון אותו למדפסת אחרת או להתקן אחר.

ניתן לכתוב device drivers להתקני תו אחרים ולשלב בהם עבודה עם תכניות פיקוח.

תכנית הפיקוח מורכבת למעשה מ-thread בעל עדיפות ביצוע גבוהה המטפל בשני מאגרים שאפשר להתייחס אליהם כמאגרי IN ומאגר OUT. היישום חייב לפתוח תחילה את התקן התו לשימוש מבוקר, והוא עושה זאת באמצעות מקשר הפיקוח שהתקבל. נתונים מההתקן מוצבים במאגר IN, וכדי לעבד אותם, ה-thread האחראי על הפיקוח קורא אותם מתוך המאגר לתוך מאגר פרטי. רק אז תכנית הפיקוח יכולה לבדוק, לשנות, להכניס ו/או למחוק תווים בנתונים שהתקבלו. אם יש נתונים שצריך להחזיר להתקן, היא כותבת את הנתונים לתוך מאגר OUT. הנתונים שנמצאים במאגר OUT מוצבים אז בזרם הנתונים הכללי להתקן. כאשר היישום רוצה לבטל את הפיקוח שלו, הוא עושה זאת בעזרת מקשר הפיקוח להתקן.

תכנית פיקוח אחת, או יותר, יכולה לשלוט על זרם הנתונים של ההתקן. הקיום של תכניות הפיקוח האחרות שקוף לחלוטין לגבי כל אחת מהן, אולם בזמן שמתבצע הרישום של התכנית, יש לה את האפשרות להצביע על ההעדפה שלה לגבי המיקום שלה ביחס לאחרות (ראשונה, אחרונה, לא חשוב) על בסיס ראשון-מגיע-ראשון-מקבל. במלים אחרות, תכנית הפיקוח הראשונה שנרשמה לעבודה וביקשה מקום ראשון תהיה ראשונה, ואילו תכנית הפיקוח הבאה שנרשמה לעבודה ובקשה מקום ראשון תהיה שניה בתור. באופן דומה, תכנית הפיקוח הראשונה שנרשמה לעבודה וביקשה מקום אחרון תהיה אחרונה, ואילו תכנית הפיקוח הבאה שנרשמה לעבודה וביקשה מקום אחרון תהיה שניה מהסוף. תכניות פיקוח שנרשמו לעבודה בלי העדפה כלשהי לגבי המיקום שלהן יוצבו

לפני אלו שביקשו להיות אחרונות.

המישקים שהיישום צריך להשתמש בהם לצורך הפיקוח רשומים להלן לפי סדר השימוש בהם.

DosMonOpen פותח את התקן התו המזוהה בעזרת שם ההתקן, ומחזיר מקשר פיקוח.

DosMonReg רושם לעבודה את תכנית הפיקוח ביישום לפי המיקום שהיא מעדיפה להיות בו ברשימת תכניות הפיקוח. היא פועלת על זרם נתונים מסוים של התקן תו המזוהה על-ידי מקשר הפיקוח שלו. תהליך יכול להפעיל תכנית פיקוח לכל זרם נתונים של ההתקן ולהשתמש לשם כך באותו מקשר פיקוח. במקרה של זרם נתונים נפרד לכל session, היישום יכול להפעיל תכנית פיקוח נפרדת לכל session.

DosMonRead קורא רשומת נתונים מתוך מאגר ה-IN לתוך שטח עבודה פרטי המוגדר על-ידי תכנית הפיקוח. במקרה שבו אין' כרגע נתונים לקריאה, היישום צריך לציין אם הוא רוצה להמתין עד אשר הוא יקבל נתונים, או שהוא רוצה שהשירות יחזור אליו מיידית.

DosMonWrite כותב רשומת נתונים מתוך שטח העבודה הפרטי לתוך מאגר OUT.

DosMonClose סוגר את ההתקן המזוהה על-ידי מקשר הפיקוח שלו, ומנתק את הקשר לזרם הנתונים של ההתקן.

ק/פ ישיר לחומרה

יישום או תת-מערכת ב-OS/2 יכולים לבצע ק/פ ישיר להתקן באמצעות סגמנטים של פקודות הרשאה (IOPL). זאת אומרת, שיישום או תת-מערכת יכולים לתכנת את המתאם להתקן מבלי להשתמש ב-device driver, או במישק בקרה מסוג IOCtl.

ישנם מספר מצבים שבהם יישום או תת-מערכת צריכים לשלוט ישירות בהתקן. לבקרה ישירה יש יתרון בכך שהיא יותר מהירה מאשר שימוש בשירותי ביניים. כדי לבצע זאת, היישום צריך להשאיר בזיכרון רק את הסגמנטים שהוא משתמש בהם בפועל. לכן, סגמנטים המכילים שגרות המשמשות לשליטה בהתקן אינם חייבים להיות בזיכרון ולתפוס בו מקום, עד אשר הם נקראים להתבצע. ב-DOS, אין למעשה הגבלות על מה שהיישום יכול לעשות עם התקן שהוא פונה אליו ישירות. עם זאת, חופש הפעולה יכול להוביל בקלות לקונפליקטים בין תכניות TSR, המתחרות ביניהן על השימוש באותו התקן. יישום ב-OS/2 יכול לפנות ישירות להתקן בתנאים מסוימים בלבד. תנאים אלה קיימים בגלל שתי סיבות: הראשונה היא, שמנגנוני ההגנה של המיקרו-מעבד 80286 מציבים הגבלות על מה שפקודה ברמת הרשאה של יישום (CPL = 3) יכולה לעשות; הסיבה השניה היא, שבסביבה רבת משימות יותר מיישום אחד יכול להשתמש בהתקן, וכדי למנוע תקלות יש צורך בכללי עבודה מסוימים.

כללי ההגנה של 80286 מונעים מפקודה ברמת הרשאה של יישום (CPL = 3) לשרת פסקי חומרה, למרות שיש צורך לטפל בהם בכל רמת הרשאה. המעבד אינו מאפשר ליישום להעלות את רמת ההרשאה שלו, ולכן רק פקודות ברמת הרשאה של

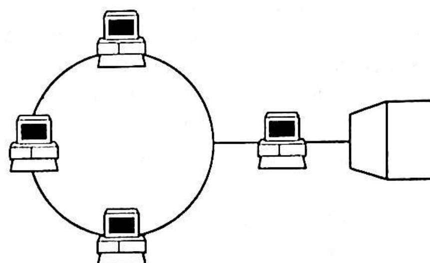
המערכת יכולות לשרת את פסקי החומרה. כתוצאה מכך, רק ה-**device driver** יכול לשרת את פסקי ההתקן, מכיון שהוא מתבצע עם רמת הרשאה מועדפת המתאימה לצורך זה. זאת אומרת, שהתקן שאינו מבוקר פסקים הינו המועמד הטוב ביותר לביצוע ק/פ ישיר. דוגמה טובה להתקן כזה הוא מתאם התצוגה.

כדי לשלוט ישירות בהתקן, היישום חייב להשתמש בפקודות ק/פ כמו IN, INS, OUT, OUTS, CLI ו-STI. הפקודות IN/INS ו-OUT/OUTS מקבלות ושולחות בהתאמה נתונים לכניסות הק/פ של ההתקן. הפקודות CLI ו-STI מנתקות ומאפשרות בהתאמה פסקים של החומרה. אולם, ה-80286 מגביל את השימוש בפקודות אלו ליישומים שיש להם רמת הרשאת ק/פ (IOPL), או רמת הרשאה גבוהה יותר. מכיון שיישומים מתבצעים ברמת ההרשאה הנמוכה ביותר, הם אינם יכולים לבצע פקודות ק/פ. OS/2 מספקת את האמצעים שבעזרתם יישום או תת-מערכת יכולים לפעול ברמת הרשאה גבוהה יותר, כדי שיוכלו לבצע ק/פ ישיר. במטרה לשמור על רמת ההגנה הקיימת, מערכת ההפעלה אינה מאפשרת לכל הסגמנטים ביישום או בתת-מערכת לפעול ברמת ההרשאה החדשה (IOPL). רק הסגמנטים שנקבע שהם זקוקים לרמת הרשאה גבוהה יותר יורשו להתבצע במסגרת IOPL. מסיבה זו, יישום או תת-מערכת חייבים לבודד את כל השגרות המשתמשות בק/פ ולהכניס אותן לתוך סגמנט פקודה אחד או יותר, ולהצביע בזמן הקישור מי מבין הסגמנטים המכילים פקודות דורש IOPL.

שירותי מערכת אינם יכולים להתבצע מתוך סגמנטים של פקודות עם הרשאת IOPL. הדבר גורם לכך, שפקודות ברמות ההרשאה השונות יכולות להגן על עצמן ביעילות זו מזו. אולם, סימון של סגמנט פקודה עם הרשאת IOPL אינו מספיק כדי לבצע פקודות ק/פ. לפני הקריאה לשיגרה מתוך סגמנט הפקודה עם IOPL, היישום או תת המערכת חייבים לבקש רשות להשתמש בהתקן, באחת משתי דרכים: האחת מאפשרת להשתמש בכניסות הק/פ של ההתקן וגם מאפשרת לטפל במצב פסקי החומרה (DosPortAccess), והאחרת מאפשרת לטפל רק במצב פסקי החומרה (DosCLIAccess).

ישנו עוד אספקט חשוב אחד הנוגע ל-IOPL. מאחר ו-IOPL מנצל את מנגנון ההגנה של המעבד, יש למשתמש הקצה שליטה על שילוב האופציה הזו. הוא עושה זאת על-ידי קביעה של מילת המפתח IOPL בקובץ ה-CONFIG.SYS. הדבר מאפשר לסגמנטים של יישום, או תת-מערכת, להתבצע ברמת הרשאה IOPL. באמצעות כלים אלה, משתמש הקצה יכול לקיים את רמת ההגנה והאמינות הדרושה במערכת. הוא יכול למנוע הרצה בשוגג של יישום אחד או יותר, אשר משנים את סביבת התפעול מבלי לדעת על כך.

רשת תקשורת מקומית



פרק 6

ניהול התקן מבוקר־פסקים

אחד המאפיינים החשובים ביותר של המחשב האישי הוא התגובה המהירה שלו לפעילויות המבוצעות על-ידי משתמש הקצה. החשובות ביניהן הן פעולות הק/פ שהמשתמש והיישומים מבצעים במטרה לטפל בנתונים. ב-DOS, פעולות ק/פ להתקן מתבצעות לפי מודל התיחקור (polling) וחייבות להסתיים לפני שהמשתמש יוכל להמשיך בפעילות כלשהי אחרת. ב-OS/2, פעולות הק/פ נבנו לפי מודל של התקן מבוקר-פסקים, אשר משפיע הן על המשתמש והן על היישומים. ניהול התקן מבוקר-פסקים מאפשר לפעולות הק/פ להתבצע בו-זמנית עם פעילויות אחרות. לכן, ביצוע רב-משימתי הוא יותר יעיל במונחים של שיתוף זמן המעבד בין פעילויות המתבצעות בו-זמנית. יעילות גבוהה יותר גורמת לתגובה מהירה יותר, שממנה נהנים הן המפעיל והן היישומים שהוא מריץ.

מדוע ניהול התקן מבוקר-פסקים הוא כה חשוב למערכת? כדי להבין את מידת ההשפעה שלו, אנו חייבים להבין תחילה את המאפיינים של ניהול התקן לפי שיטת התיחקור.

כפי שתזכור, ביצוע חד-משימתי ב-DOS אומר, שאין אף מנגנון ב-DOS שיכול להודיע ליישום שבקשת הק/פ שלו הסתיימה. יישום ב-DOS (וכתוצאה מכך גם משתמש ב-DOS) חייב לבצע את הפעולות שלו בצורה סידרתית. דרישה זו מכתיבה את העקרון הבסיסי של ניהול ק/פ תחת DOS בשיטת התיחקור. במלים אחרות, התכנית שהוציאה בקשת ק/פ להתקן, מבצעת לולאה של הוראות הבודקות אם חל שינוי במצב של סמן מסוים. אם מצב הסמן השתנה, התכנית יודעת שפעולת הק/פ הסתיימה, ואם מצב הסמן לא השתנה, התכנית חייבת לבצע שוב את לולאת ההוראות. הסמן יכול להיות תוצאה של שינוי בערך הנמצא באחד מחיבורי הק/פ (ports) של ההתקן, או תוצאה של פסק בהתקן. במקרה האחרון, הפסק המופיע בהתקן מפעיל שיגרה לטיפול בפסקים, אשר משנה את מצב הסמן שהתכנית בודקת בלולאה.

מדוע חשוב לדעת שפעולת הק/פ הסתיימה? סיבה אחת היא טיפול בשגיאות. אם פעולת הק/פ נכשלת, יישום ה-DOS צריך לנסות שוב את הפעולה, או להודיע למפעיל שיש צורך להפסיק את הפעילות הזאת בגלל שגיאה. הצלחה בביצוע הק/פ אומרת שיישום DOS יכול להמשיך לפעולה הבאה. סיבה שניה היא ניצול של הזיכרון. לאחר שפעולת הק/פ הסתיימה, היישום יודע שהמאגרים המכילים את הנתונים לצורך פעולת הק/פ פנויים וניתנים לשימוש חוזר.

תחת DOS, יישום אינו נדרש לנהל את התיחקור. אם יישום DOS משתמש במישקי המערכת לצורך ביצוע ק/פ, מישקים אלה מנהלים את התיחקור לגבי מצב פעולת הק/פ. אבל יישום DOS המטפל ישירות בהתקנים, חייב לנהל את התיחקור בעצמו. בכל מקרה, ניהול התקנים תחת DOS מוגבל לשימוש סידרתי ולתיחקור בלבד.

ניהול מבוקר-פסקים של התקן ב-OS/2 משחרר את ההגבלות שהיו על ק/פ בשיטת התיחקור. ניהול מבוקר-פסקים של ההתקן מאפשר למבקש הק/פ להודיע על סיום הפעולה. בזמן ביצוע הק/פ בהתקן ניתן לבצע פעילות אחרת, ולמעשה גם סוגים שונים של ק/פ יכולים להתבצע בו-זמנית, כמו למשל ק/פ לדיסק, למדפסת, למסך ולעכבר. כתוצאה מכך משאבי המערכת יכולים להיות מנוצלים בצורה יעילה על-ידי מספר יישומים. הדבר נכון גם לגבי יישום אחד שבו פועלים מספר תהליכים או threads. ביישום כזה יכולה להתבצע חפיפה בין פעולות עיבוד לבין בקשות ק/פ.

יתרון נוסף של ניהול מבוקר-פסקים של התקן בסביבה רבת משימות הוא השימוש האופטימלי בהתקנים "חכמים". המעבד הראשי יכול להעביר למעבדים בהתקנים אלה עבודה המתאימה להם, ובכך להשתחרר לשם טיפול ביותר פעילויות הקשורות ביישום. כתוצאה מכך ניתן להגיד ש-OS/2 מתאימה היום לפיתוח עתידי של התקנים.

הבסיס לניהול מבוקר-פסקים של התקן הינו החיבור בין מערכת ההפעלה לבין התכנית השולטת על ההתקן, במקרה שלנו ה-device driver. ה-thread של ה-device driver שהוציא את בקשת הק/פ מוותר על פלח הזמן שלו בשעה שהוא ממתין להתקן שיסיים את פעולתו. כאשר ההתקן מוכן, הוא שולח פסק למערכת. הפסק פורץ במהלך הביצוע של ה-thread הפעיל, וגורם להפעלת שיגרת הטיפול בפסקים של ה-device driver. שיגרה זו מאתחלת את מצב הפסק בהתקן ומודיעה למערכת ההפעלה שה-thread של ה-device driver שהמתין לסיום בקשת הק/פ מוכן לריצה. לאחר ששיגרת הטיפול בפסקים סיימה את פעולתה, יכולה מערכת ההפעלה לתזמן את ה-threads השונים בהתאם לעדיפות הביצוע שלהם, ולהפעיל את ה-thread בעל עדיפות הביצוע הגבוהה ביותר. התזמון הזה חשוב בעיקר לפעילויות שגורם הזמן בהן הינו חשוב ביותר, כי המשך פעולתן תלוי במצב התקן הק/פ. אחת הדוגמאות הינה תקשורת נתונים, שבה נדרשות פעילויות מסוימות כדי לתפעל את ערוץ התקשורת. כאשר ה-thread של ה-device driver מתבצע, הוא מחזיר קוד סיום ליישום שיזם את בקשת הק/פ.

כפי שרואים, ניהול מבוקר-פסקים של התקן משתלב בצורה מלאה עם הביצוע הרב-משימתי שמכתיבה מערכת ההפעלה, והוא גם מגיב מהר יותר למצבים השונים בהתקן. התגובה המהירה מושגת מכיון שההתקן מאותת על מצבו בעזרת שידור של פסק בחומרה.

יישום הפועל תחת OS/2 יכול לבצע ק/פ בשיטת התיחקור. אולם, חזרה ללא צורך על הוראות של לולאה הינה ניצול לא יעיל של זמן הביצוע המוקצה לו. כדי לעקוף זאת אפשר להשתמש במישקים המספקים שירותים של השעון הפנימי כמו DosTimerStart, או DosSleep, כדי לקבוע את התדירות שבה היישום יבדוק אם ההתקן סיים את פעולת הק/פ שלו.

תפקיד ה-DEVICE DRIVERS

מודל ההתקן מבוקר הפסקים ב-OS/2 מושג על-ידי שימוש ב-device driver. ב-OS/2, רק ה-device driver יכול לטפל בפסקים של התקנים. לכן, יישום שצריך לנהל התקן מבוקר-פסקים חייב לספק device driver משלו, או להשתמש בזה הקיים לאותו התקן.

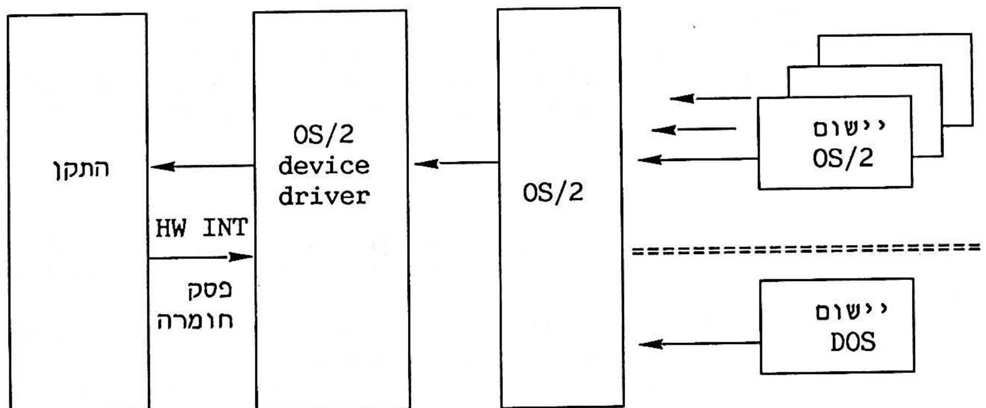
ההגדרה של device driver ב-OS/2 דומה לזו שב-DOS. זוהי תכנית שמנהלת את זרימת הנתונים להתקן וממנו. היא אינה יישות עצמאית בדומה לתהליך, אלא דומה יותר לתת-שיגרה. מערכת ההפעלה קוראת ל-device driver כדי שיבצע פעולה מסוימת: בשמה של בקשת ק/פ של היישום ולמעשה, אפשר להסתכל עליו כהרחבה של מערכת ההפעלה.

בקשות הק/פ של היישום ב-OS/2 מופיעות בשתי צורות. צורה אחת נגזרת כתוצאה מהשימוש במצב המוגן של ה-80286, והצורה השניה נגזרת כתוצאה מהשימוש במצב האמיתי של המעבד. כתוצאה מכך, מערכת ההפעלה קוראת לאותו device driver שיבצע את בקשת הק/פ גם במצב מוגן וגם במצב אמיתי (ראה תרשים 44). פסקים בחומרה ימשיכו להתרחש בלי תלות במצב שבו מופעל ה-CPU. לכן אפשר להגיד, שה-device driver ב-OS/2 יכול להתבצע בסביבה דו-מצבית, כלומר גם במצב האמיתי של ה-80286 וגם במצב המוגן.

במצב מוגן, ה-device driver מתבצע עם רמת הרשאה של המערכת, כלומר רמת הרשאה=0, אשר מעניקה לו רמת-הרשאה לק/פ (IOPL). IOPL מאפשר ל-device driver לגשת לכניסות הק/פ של ההתקן ולבצע פקודות מוגנות כמו IN ו-OUT. ה-device driver יכול גם לטפל במצב אות-הסימון של הפסק, ובכך לאפשר או לבטל את השימוש בפסקים. יישום ב-OS/2 יכול לגשת לכניסות הק/פ של ההתקן בעזרת שגרות המקובצות בסגמנטים מכילי פקודות עם הרשאת IOPL. יש להדגיש שרק ה-device driver יכול לטפל בפסקי חומרה.

התקנת device driver ב-OS/2 דומה להתקנה ב-DOS, באמצעות המשפט =DEVICE בקובץ התצורה CONFIG.SYS. בזמן התיחול של מערכת ההפעלה, נבדק קובץ התצורה ואז הקובץ המצוין על-ידי =DEVICE נטען לזיכרון. לאחר הטעינה קוראת מערכת ההפעלה לתכנית ה-device driver לתחל את עצמה.

במערכת ישנם מספר device drivers, אשר חלקם נטענים אוטומטית (המסך, המקלדת, המדפסת, דיסק/דיסקט והשעון), ואת השאר יש צורך להטעין בעזרת המשפט =DEVICE שבקובץ התצורה (העכבר, תקשורת אסינכרונית ודיסק בפועל). על-ידי כך שסיפקנו מערכת בסיסית של device drivers איפשרנו ל-OS/2 להבטיח תמיכה בתצורה של מערכת המכילה התקנים ופונקציות סטנדרטיות.



תרשים 44. המקום של device driver במערכת

היכולת להתקין device drivers מאפשרת להרחיב את התיפקודיות של מערכת ההפעלה. ניתן להחליף device driver קיים כדי לספק פונקציה נוספת להתקן, ואפשר גם להוסיף device driver חדש למערכת כדי לתמוך בהתקן חדש.

כמו ב-DOS, יש שני סוגים של device drivers ב-OS/2: האחד - להתקנים המטפלים בתווים (להלן התקני תווים) והשני - להתקנים המטפלים בבלוקים (להלן התקני בלוקים).

התקן תו מבצע ק/פ של תו אחד בו-זמנית, בד"כ דרך פקודות ק/פ בתכנית (IN ו-OUT בכניסות הק/פ). בהתקני תו, סדר הופעת התווים הוא משמעותי, כי ה-device driver של התקן התו אינו יכול להכנס לנתונים בצורה אקראית. סדר בקשות הק/פ גם הוא משמעותי, כי ה-device driver ימלא אחר הבקשות לפי הסדר שהוא קיבל אותן. הסיבה העיקרית שבגללה הסדר הוא כל-כך חשוב בהתקני תו היא, שלהתקני תו אין זיכרון שבתוכו הם יכולים לשמור את הנתונים העוברים דרכם. לאחר שמידע נכתב או נקרא, אין אפשרות לאחזר אותו. התקן תו ניתן לזיהוי והפניה אליו נעשית בעזרת שם ההתקן. זיהוי ההתקן בעזרת שם מאפשר ליישום להתייחס להתקן כקובץ, ולהשתמש בפונקציות מסוימות של מערכת הקבצים כדי לבצע בו פעולות ק/פ. דוגמאות להתקני תו ושמותיהם: מדפסת (LPT1, LPT2), מקלדת (KBD$) ותקשורת אסינכרונית (COM1, COM2).

התקן בלוקים מבצע בד"כ ק/פ עם בלוקים של נתונים מהדיסק, באמצעות אביזר DMA (גישה ישירה לזיכרון). המונח השימושי לבלוק נתונים הוא סקטור. סקטור הוא יחידת אחסון בדיסק בגודל 512 בתים, אשר ה-device driver יכול להכנס אליה בכל זמן. התקן בלוקים אינו דורש שהק/פ לסקטורים יבוצע באותו סדר שבו הסקטורים ממוקמים בהתקן. זאת אומרת, שה-device driver של התקן בלוקים יכול לסדר מחדש בקשות לק/פ שהוא מקבל במטרה לבצע אופטימיזיציה של ביצועי ההתקן. לדוגמה, הוא יכול למיין את הבקשות לפי המספר היחסי של הסקטור, כדי לצמצם את בזבוז הזמן שנגרם כתוצאה מהמתנה להתקן שימצא את מיקום הסקטור. סדר הסקטורים אינו חשוב להתקן הבלוקים, כי התקן זה הוא התקן אחסון, השומר את הנתונים שהוא מקבל. התקן בלוקים ניתן לזיהוי והפניה אליו נעשית בעזרת אות הכונן. יישום אינו ניגש בד"כ ישירות להתקן בלוקים, אלא מבצע ק/פ לקובץ, המהווה ייצוג לוגי של נתונים בסקטור. הוא גם משתמש באות הכונן כדי לציין באיזה התקן בלוקים הקובץ נמצא. לאחר מכן מערכת הקבצים ממירה את בקשת הק/פ לקובץ לבקשת ק/פ לסקטור, אשר נשלחת ל-device driver. ההתקנים הבאים הינם התקני בלוקים: כונני דיסקים קשיחים, כונני דיסקטים ודיסקים בפועל. השמות שלהם יכולים להיות :a ו-:b וכו'.

DEVICE DRIVERS והק/פ של היישומים

ניהול ההתקנים נעשה על-ידי device drivers בהקשר לק/פ של היישומים. יישומי OS/2 מבצעים ק/פ עם המישקים הבאים:

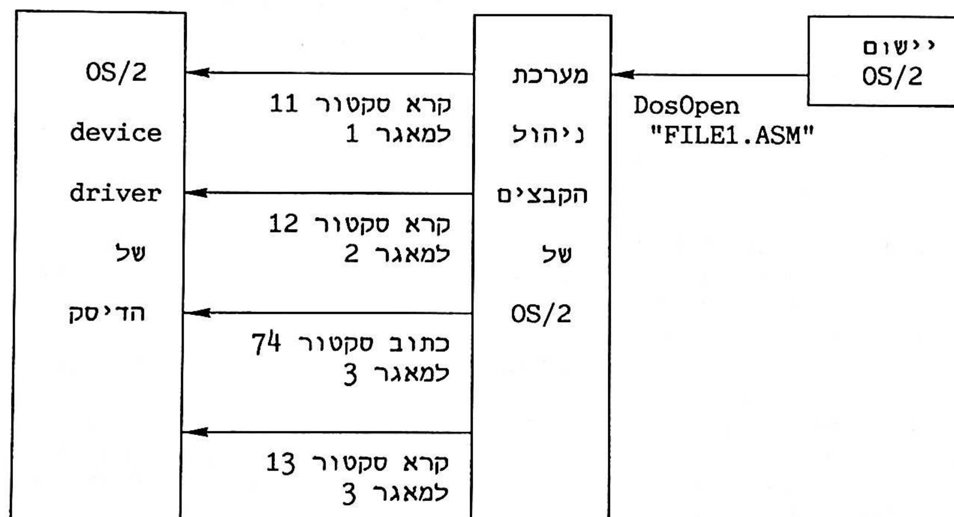
- מישקי מערכת הקבצים.
- מישקי ה-IOCtl.
- מישקי תת-המערכות לק/פ.
- מישקי פיקוח על התקני תו.

יישומים משתמשים במישקי מערכת הקבצים, כדי לבצע ק/פ לאובייקטים המבוססים על שמות (קבצים) ואשר הגישה אליהם נעשית בעיקר באמצעות device drivers המטפלים בהתקני בלוקים בלבד. מישקי מערכת הקבצים מאפשרים ליישום לסקור את הנתונים שבהתקן בצורה של מבנים לוגיים. קובץ הוא מבנה נתונים לוגי הנמצא בהתקן בלוקים שאפשר לראות אותו כמחרוזת של בתים. היישום מבצע ק/פ לקובץ על-ידי כך שהוא מזהה את המיקום היחסי של הבית הראשון בקובץ ואת מספר הבתים שיש להעביר. אולם, ה-device driver של התקן הבלוקים מבצע ק/פ רק במונחים של סקטורים. יותר מכך, הוא אינו יכול לזהות איזה סקטורים מכילים את הנתונים של היישום. מערכת הקבצים הופכת את הבתים המתייחסים לבקשת הק/פ של היישום לסקטורים בהתקן המתאימים לקובץ. בעבור כל בקשת ק/פ של היישום, מערכת הקבצים שולחת בקשת ק/פ (המבוססת על סקטורים) אחת או יותר ל-device driver של התקן הבלוקים, והוא מעביר את הסקטורים המתאימים לבקשה.

לדוגמה, יישום המשתמש במישקי הק/פ פותח (OPEN), תחילה את הקובץ, לאחר מכן הוא קורא (READ) או כותב (WRITE) לקובץ, ולבסוף סוגר (CLOSE) אותו. כתוצאה מבקשת היישום להקים קשר עם ההתקן (לפתוח את הקובץ), מערכת הקבצים יכולה לבקש מה-device driver לקרוא סקטורים, המייצגים את המדריך ומכילים מידע אחר המאפשר בקרה על ההתקן. מערכת הקבצים יכולה גם לבקש מה-device driver לכתוב סקטורים להתקן במטרה לרוקן buffers פנימיים של מערכת הקבצים (ראה דוגמה בתרשים 45). כתוצאה מביצוע ק/פ (READ או WRITE) על-ידי היישום, מערכת הקבצים מודיעה ל-device driver של התקן הבלוקים לכתוב או לקרוא סקטורים. כתוצאה מכך שהיישום ניתק את הקשר להתקן (סגר את הקובץ), מערכת הקבצים מבקשת מה-device driver לכתוב להתקן סקטורים המייצגים את המדריך המעודכן ומידע אחר המאפשר בקרה על ההתקן. ה-device driver אינו רואה את סוג הבקשה שהיישום ביקש, אלא הוא פועל לפי הנחיות מערכת הקבצים, הפועלת בשמן של בקשות הק/פ של היישום.

ניתן להשתמש במספר מישקים של מערכת הקבצים לצורך ק/פ עם התקני תו, מאחר והתקנים אלה הם בעלי שמות. מדובר בעיקר ב-OPEN, CLOSE, READ ו-WRITE. היישום יכול לראות את הנתונים של התקן התו רק בצורה של מבנים לוגיים בדומה להתקני בלוקים; כלומר, כמחרוזות של בתים (או תווים). היישום מבצע ק/פ להתקן זה על-ידי ציון ההתחלה של מחרוזת הנתונים ומספר הבתים שיש להעביר. ה-device driver יכול למלא אחר בקשת הק/פ של היישום ללא עזרה של מתווך. ואמנם, מערכת הקבצים תשלח אליו, במקרה זה, את בקשות הק/פ של היישום מבלי לשנות דבר.

לדוגמה, יישום המשתמש במישקי הק/פ לקובץ פותח (OPEN) תחילה את שם ההתקן, לאחר מכן קורא (READ) או כותב (WRITE) להתקן, ולבסוף סוגר (CLOSE) אותו. כתוצאה מבקשת היישום להקים קשר עם ההתקן (לפתוח את ההתקן), מערכת הקבצים מבקשת מה-device driver להקים קשר בין שניהם, ולהכין את ההתקן לשימוש. כתוצאה מביצוע ק/פ (READ או WRITE) על-ידי היישום, מערכת הקבצים מבקשת מה-device driver של התקן התווים לכתוב או לקרוא בהתאמה (ראה גם תרשים 46). כתוצאה מכך שהיישום ניתק את הקשר להתקן (סגר את ההתקן), מערכת הקבצים מודיעה ל-device driver לבטל את הקשר הקיים להתקן, להפסיק את הק/פ ולאתחל משתנים פנימיים. אפשר לומר שה-device driver של התקן התווים רואה את סוג בקשת הק/פ המגיעה מהיישום, כתוצאה מפעולת הניתוב של מערכת הקבצים.

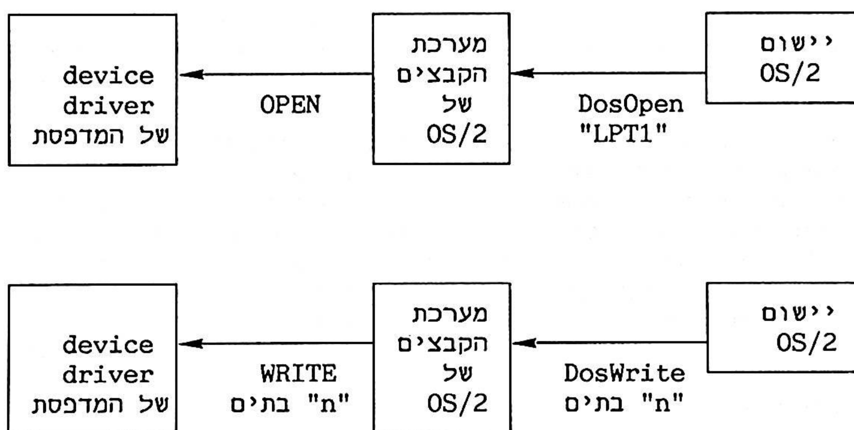


תרשים 45. דוגמה לק/פ מנקודת המבט של Block Device Driver

מישק ה-IOCtl, או מישק לבקרת ק/פ, משמש למשלוח פקודות ייחודיות להתקן, אשר עוזרות בבקרה ובשליטה עליו. מישק ה-IOCtl יכול לפעול עם device drivers המשמשים הן את התקני התווים והן את התקני הבלוקים. כאשר יישום מוציא בקשת IOCtl, מערכת ההפעלה שולחת את הבקשה ישירות ל-device driver המתאים.

לדוגמה, המשתמש במישק ה-IOCtl פותח תחילה את שם ההתקן בעזרת מישק של מערכת הקבצים. לאחר מכן הוא שולח בקשת IOCtl ולבסוף סוגר את ההתקן בעזרת מישק של מערכת הקבצים. הפתיחה והסגירה קובעות את מצב החיבור בין היישום להתקן, והן מטופלות על-ידי מערכת הקבצים בהתאם לסוג ההתקן. לאחר הפתיחה נשלחת בקשת ה-IOCtl ישירות מן היישום אל ה-device driver המזוהה על-ידי החיבור בין היישום להתקן. פקודת IOCtl תגיע ישירות מן היישום אל התקני תווים או אל התקני בלוקים.

מישקי תת-מערכות הק/פ משמשים לשירותי ק/פ המותאמים לאופי של ההתקן: תצוגה (VIO), מקלדת (KBD) ועכבר (MOU). הן מספקות מישקים יחודיים לכל אחד מהתקני התו האלה. אפשר לראות את מערכת הקבצים כתת-מערכת ק/פ המשמשת לשירותים הקשורים בהתקני בלוקים. מישקי תת-מערכת הק/פ מאפשרים ליישום לראות את הנתונים הנמצאים בהתקן רק בצורה של מבנים לוגיים המותאמים לאופי של ההתקן. לדוגמה, אפשר לטפל בנתונים הקשורים לתצוגה במונחים של תווים ומאפייני התצוגה שלהם, או במונחים של מאגר תצוגה לוגי. אפשר להתייחס לנתוני המדפסת והעכבר במונחים של רשומות, המכילות מידע על נתוני ההתקן. כאשר היישום מבצע בקשת ק/פ השייכת לתת-מערכת, תת-מערכת הק/פ יכולה להשתמש בשגרות IOPL, במישקי מערכת הקבצים או בבקשות ה-IOCtl כדי לטפל בק/פ. אם תת-מערכת הק/פ משתמשת במישקי מערכת הקבצים או במישק ה-IOCtl, היא צריכה גם להפיק בקשות ל-device driver המתאים. והיא אמנם מפיקה בקשה אחת, או יותר, כדי למלא אחר בקשת היישום.



תרשים 46. ק/פ מנקודת המבט של device driver להתקני תו

מישק תכנית הפיקוח להתקני תו משמש לשליטה בזרם הנתונים אל ההתקן וממנו. מישק זה ישים רק לגבי device drivers התומכים בתכניות פיקוח. מבין ה-device drivers הבסיסיים המסופקים עם OS/2, רק ה-device drivers של ההתקנים הבאים תומכים בתכניות פיקוח להתקנים שלהם: מדפסת, מקלדת ועכבר. היישום המבצע את תכנית הפיקוח יכול לסנן את נתוני ההתקן הנמצאים בזרם הנתונים, להשתמש בהם או להחליף אותם. ה-device driver מעביר אל תכנית הפיקוח במקרה זה כל נתון שהוא מקבל, והיא שולחת אותו ליעדו האחרון.

היחס בין הסוגים השונים של בקשות הק/פ לבין מה שה-device driver נדרש לעשות תלוי ברכיב המערכת האחראי לביצוע מישק הק/פ. הרכיב הזה יכול לבצע מספר פעולות ק/פ כדי למלא אחר בקשת ק/פ אחת של היישום. מנקודת המבט של ה-device driver, לא תמיד אפשר לדעת את הסוג המדויק של בקשת הק/פ המבוצעת על-ידי היישום. רק במקומות שבהם ה-device driver קשור בצורה הדוקה לתת-מערכת או למישק, הוא יוכל לזהות את המצב המדויק של של היישום המבקש. ה-device driver של התקן הבלוקים אינו צריך לדעת את סוג בקשת הק/פ, כי מערכת הקבצים דואגת לחיבור היישום להתקן לצורך ביצוע ק/פ. לעומת זאת, device driver להתקני תו צריך לעתים קרובות לנהל את החיבור בין היישום לבין התקן התו, ובעיקר במקרים שבהם אין תת-מערכת שתבצע את הפונקציה הזו. במקרה כזה, ה-device driver של התקן התו יכול לבקש ממערכת הקבצים, באמצעות הכותרת (header) של ההתקן, להודיע לו כאשר היישום מקים או מבטל את הקשר שלו להתקן התו. במלים אחרות, כאשר היישום פותח את ההתקן, מערכת הקבצים שולחת פקודת OPEN ל-device driver וכאשר היישום סוגר את ההתקן, מערכת הקבצים שולחת פקודת CLOSE. פעולה זו מאפשרת ל-device driver לצרך בקשות ק/פ (read & write) עוקבות המתייחסות לחיבור מסוים. במקרים כאלה ישנה אפשרות שה-device driver יצטרך גם לתחל (initialize) את מצב ההתקן בכל פעם שיישום מתחבר לראשונה להתקן.

המישק שרכיבי המערכת, כמו מערכת הקבצים, משתמשות בו כדי להודיע ל-device driver מה לעשות, נקרא מישק חבילת הבקשה (request packet). הרכיב במערכת ההפעלה קורא ל-device driver ומעביר לו מחוון לחבילת

הבקשה, שאינה אלא מבנה נתונים (ראה תרשים 47). מבנה זה מתחיל בכותרת באורך קבוע המכילה מידע המשותף לכל הבקשות הנשלחות ל-device driver, ולאחריה ישנו שטח באורך משתנה המכיל מידע ייחודי לבקשה מסוימת.

הכותרת מכילה חמישה שדות חשובים: אורך, יחידה, פקודה, מצב וקישור. שדה האורך מודיע ל-device driver על הגודל האמיתי של החבילה. מאחר וישנו שטח בעל אורך משתנה, הגודל של המבנה אינו זהה בכל בקשה. שדה היחידה מתייחס רק ל-device drivers המטפלים בבלוקים, והוא מזהה את התקני הבלוקים, מבין שאר ההתקנים הנמצאים בבעלות ה-device driver, שהינו היעד של הבקשה. שדה הפקודה מודיע ל-device driver על הפעולה שיש לבצע. שדה המצב ממולא על-ידי ה-device driver לאחר שהוא ביצע את הבקשה, ובו מוצב הקוד החוזר ליישום. שדה הקישור יכול לשמש את ה-device driver כדי לאחסן את רשימת החבילות שיש לקשר. device driver צריך לשמור רשימה של בקשות שלא הושלמו במשך אותן פעמים שההתקן עסוק ואינו יכול לעבוד על בקשה אחרת.

0	אורך	יחידה	פקודה	מצב
5	שמור			
9	קישור			אחר

	שדה	רוחב	משמעות
length	אורך	1 בית	האורך הכללי של מבנה הנתונים
unit	יחידה	1 בית	איזה התקן בלוקים
command	פקודה	1 בית	הפעולה שיש לבצע
status	מצב	2 בתים	קוד הסיום של הפעולה
reserved	שמור	4 בתים	
linkage	קישור	4 בתים	רשימת החבילות שיש לקשר
other	אחר	משתנה	מידע על פקודה מיוחדת

תרשים 47. מבנה חבילת הבקשה

פקודות להתקני בלוקים

הפקודות שה-device driver המטפל בהתקני בלוקים יכול לקבל:

- Initialize
- Media Check
- Build BIOS Parameter Block
- Read
- Write
- Write With Verify
- Removable Media Support
- Generic IOCtl
- Reset Media
- Get Logical Drive Map
- Set Logical Drive Map
- Query Partitionable Fixed Disks
- Get Fixed Disk/Logical Units

השדה "יחידה" בכותרת חבילות הבקשה לפקודות אלו, מזהה את התקן המטרה. הערך המוצב בשדה "יחידה" הוא יחסי למספר הכללי של ההתקנים הנתמכים על-ידי ה-device driver: הוא נע בין 0 ל-n-1, כאשר n הוא מספר ההתקנים הלוגיים הנתמכים. device driver להתקן בלוקים התומך בארבעה התקנים (שני כונני דיסקטים ודיסק קשיח שיש בו שתי מחיצות) יראה שערך זה נע בין 0 ל-3.

לגבי התקני בלוקים, מערכת ההפעלה משתמשת גם בתפיסה של מיפוי התקן לוגי להתקן פיסי. אם קיים רק כונן דיסקטים יחיד שיכול לטפל בשתי אותיות הכונן "a:" ו-"b:", אפשר לקבוע שההתקן הפיסי יוכל לייצג רק אחד משני ההתקנים הלוגיים בו-זמנית. במקרה שבו נקבעו מחיצות בדיסק הקשיח, ההתקן הפיסי מחולק בין מספר התקנים לוגיים, ולכל התקן כזה יש אות כונן משלו.

בתגובה לפקודה **Initialize**, ה-device driver צריך לכוון את ההתקנים שלו ולהחזיר בחבילת הבקשה את מספר ההתקנים הלוגיים שבהם הוא תומך. לדוגמה, במערכת שבה יש כונן דיסקטים יחיד, ה-device driver יחזיר את הערך 2, המצביע על תמיכה בכוננים "a:" ו-"b:". הוא חייב להחזיר גם את הסוגים של התקני הבלוקים בהם הוא תומך. הוא עושה זאת על-ידי העברת מערך של BIOS Parameter Blocks (BPB להלן), אשר מתאר את המאפיינים של התקן הבלוקים הכוללים את מספר הראשים, מספר המסילות ומספר הסקטורים בכל מסילה. פקודה זו נשלחת ל-device driver אשר התכנית שלו נטענת על-ידי השורה =DEVICE בקובץ ה-CONFIG.SYS, ואם ישנם משתנים כלשהם בשורת =DEVICE, הם מועברים אליו.

מערכת הקבצים משתמשת בפקודה **Media Check** בעיקר כדי לקבוע אם השתנה המצע המגנטי בהתקן הבלוקים. ה-device driver חייב להצביע אם המצע השתנה או לא. אולם, לגבי מספר התקני בלוקים, אין ה-device drivers שלהם יכולים לגלות שינוי שבוצע בידי המשתמש, ובמקרה זה עליהם להגיב בתשובה של אי-ודאות לגבי המצע המגנטי הנמצא בהתקן. לגבי התקני בלוקים אלה, ה-device driver חייב להגדיר נקודת זמן שמעבר לה הוא לא יהיה בטוח לגבי המצע בהתקן. בד"כ, device drivers שאינם יכולים לגלות שינוי במצע המגנטי יערכו מעקב אחר פרק הזמן שעובר בין בקשות ק/פ להתקן שמדובר בו. כאשר פרק הזמן ארוך, אפשר להניח שהחליפו את המצע. כמובן

שהמצע אינו יכול לחשתנות אם ההתקן הוא דיסק קשיח.

מערכת הקבצים משתמשת בפקודה **Build BIOS Parameter Block** בכל פעם שה-device driver מודיע שהמצע המגנטי השתנה או שיתכן והשתנה. לאחר שהוא קיבל פקודה זו, הוא חייב לזהות את המצע בהתקן ולהחזיר BPB ומתאר של המצע.

הפקודות **Read**, **Write**, **Write With Verify** מודיעות ל-device driver להעביר מספר סקטורים, המתחילים בסקטור מסוים. ההעברה נעשית מ-buffer שצוין על-ידי מערכת הקבצים מצע המגנטי בהתקן ולהיפך. לאחר שה-device driver מלא את הבקשה, הוא חייב לציין את מספר הסקטורים שהועברו בפועל במסגרת חבילת הבקשה. אולם, אם הוא גילה שהמצע המגנטי השתנה או שיתכן והשתנה, הוא חייב להחזיר שגיאה של "Uncertain Media" לבקשת הק/פ ולכל בקשות הק/פ שיבואו אחריה, עד אשר מערכת הקבצים תאתחל את ההתקן (Reset Media).

הפקודה **Removable Media Support** מופקת כתוצאה משימוש של היישום ב-IOCtl. בעזרת פקודה זו היישום מברר אם המצע המגנטי הוא נתיק או לא (כלומר, דיסק קשיח מול דיסקט). ה-device driver מגיב על-ידי הצבת סימן בחבילת הבקשה.

הפקודה **Generic IOCtl** נשלחת ל-device driver כתוצאה מבקשת IOCtl שנעשתה על-ידי היישום (הפונקציה DosDevIOCtl, או הפונקציה Int 21h). ה-device driver בודק את הקטגוריה של הפונקציה ואת קוד פונקציית המשנה, כדי לקבוע איזה פעולה הוא צריך לבצע. ישנן שתי קטגוריות המתייחסות במיוחד להתקני בלוקים: קטגוריה 8 שהיא בקרת דיסק לוגי, וקטגוריה 9 שהיא בקרת דיסק פיסי. ההבדל בין שתי הקטגוריות הוא, שפקודות ה-IOCtl הנמצאות בקטגוריה 8 מתייחסות להתקן לוגי (או למחיצה בהתקן פיסי), ואילו פקודות ה-IOCtl הנמצאות בקטגוריה 9 מתייחסות להתקנים פיסיים הניתנים לחציצה. בפיסקאות הבאות תמצא תיאור קצר של פונקציות המשנה שה-device driver יכול לקבל. ראוי לדעת שישנן פונקציות משנה נוספות, אשר מיורשות על-ידי מערכת ההפעלה ואינן מועברות ל-device driver. פונקציות אלה הן מחוץ לתחום הנושאים שבהם עוסק ספר זה.

קטגוריה		תיאור
8		**בקרה של דיסק לוגי**
	43h/63h	קבע/השג פרמטרים של ההתקן. מודיע ל-device driver לקבוע או להשיג BPB להתקן הבלוקים, או למצע המגנטי בתוך ההתקן.
	44h/64h	כתוב/קרא מסילה. מודיע ל-device driver לבצע ק/פ על סקטורים במסילה בעזרת טבלת המסילה המועברת בחבילת הבקשה.
	45h/65h	פרמט ואמת מסילה/אמת מסילה. מודיע ל-device driver לכוון ו/או לבדוק את המבנה של המסילה, בהתאם לטבלת המסילה המועברת בחבילת הבקשה.

קטגוריה	תיאור
9	**בקרה של דיסק פיסי**
44h/64h	כתוב/קרא מסילה פיסית. מודיע ל-device driver לבצע ק/פ על סקטורים במסילה בעזרת טבלת המסילה המועברת בחבילת הבקשה. מספר המסילה ניתן באופן יחסי להתחלה של כונן פיסי ולא של כונן לוגי.
63h	השג פרמטרים של ההתקן הפיסי. מודיע ל-device driver להחזיר מידע על ההתקן הפיסי.
65h	אמת מסילה פיסית. מודיע ל-device driver לבדוק את המסילה בעזרת טבלת המסילה המועברת בחבילת הבקשה. מספר המסילה ניתן באופן יחסי להתחלה של כונן פיסי ולא של כונן לוגי.

מערכת הקבצים משתמשת בפקודה **Reset Media** כדי לאשר את קבלת קוד השגיאה "Uncertain Media" שה-device driver החזיר על ביצוע בקשה קודמת להתקן הבלוקים. היא מאפשרת לו לדעת שהוא אינו צריך להחזיר קוד שגיאה לגבי המצע המגנטי הנמצא בהתקן הבלוקים.

המערכת משתמשת בפקודות **Set Logical Drive Map** ו-**Get Logical Drive Map** כדי להודיע ל-device driver להגדיר התקן לוגי מסוים להתקן פיסי. זוג הפקודות הזה משמש בעיקר במערכות שבהן יש רק כונן דיסקטים אחד שתומך באותיות הכונן "a:" ו-"b:". במקרה זה, שני ההתקנים הלוגיים שייכים להתקן פיסי אחד. פקודות אלו גם מאפשרות ל-device driver ולמערכת ההפעלה להשתמש בכונן אחד, כך שיתמוך במצעים עם מאפייני פירמוט אחרים. לדוגמה, הפקודה Get Logical Drive Map מבקשת מה-device driver לזהות איזה התקן לוגי מחובר כרגע להתקן הפיסי. הפקודה Set Logical Drive Map מודיעה לו לחבר את ההתקן הפיסי להתקן הלוגי שצוין בפקודה.

המערכת משתמשת בפקודה **Query Partitionable Fixed Disks** כדי לזהות את מספר התקני הבלוקים הניתנים לחציצה (כלומר דיסקים קשיחים) הנמצאים בבעלות ה-device driver. ספירה זו היא של התקנים פיסיים, ולא של מספר המחיצות הקיימות בדיסקים הקשיחים.

המערכת משתמשת בפקודה **Get Fixed Disk/Logical Units** כדי לזהות איזה התקנים לוגיים קיימים בהתקן בלוקים מסוים ובמלים אחרות, כמה מחיצות קיימות בדיסק קשיח. ה-device driver מחזיר בחבילת הבקשה, מפת סיביות, אשר מייצגת את ההתקנים הלוגיים הנמצאים בבעלותו, והוא חייב לקבוע את הסיבית המתאימה להתקן הלוגי. לדוגמה, device driver תומך בשני כונני דיסקטים ודיסק קשיח, אשר מחולק לארבע מחיצות ולכן עליו לקבוע את מפת הסיביות, כך שתצביע על-כך שיחידות 2 עד 5 קיימות בדיסק הקשיח.

פקודות של התקני תו

device driver המטפל בהתקני תו יכול לקבל את הפקודות הבאות:

- Initialize
- Read
- Peek
- Input Status
- Input Flush
- Write
- Output Status
- Output Flush
- Device Open
- Device Close
- Generic IOCtl
- DeInstall

בחבילת הבקשה לפקודות אלו, אין שדה המזהה את התקן המטרה שאליה מיועדת חבילת הבקשה. ה-device driver של התקני תו משתמש במנגנון שונה מזה של התקני הבלוקים, לזיהוי ההתקנים שלו, ולכן הוא חייב להגדיר מבנה נתונים מיוחד, הנקרא כותרת ההתקן (Device Header), עבור כל התקן הנמצא בבעלותו. כל כותרת התקן חייבת להכיל נקודת כניסה ייחודית לתוך ה-device driver. ואז, כאשר ה-device driver נקרא באמצעות חבילת הבקשה, הוא ייקרא בנקודת הכניסה הקשורה להתקן המטרה.

הפקודה **Initialize** מודיעה ל-device driver שהוא צריך לכוון את ההתקן לצורך פעולות ק/פ עתידיות. פקודה זו נשלחת, כאשר תכנית device driver נטענת על-ידי השורה =DEVICE בקובץ ה-CONFIG.SYS. ואם ישנם משתנים כלשהם בשורת =DEVICE, הם מועברים גם-כן.

פקודות **Read** ו-**Write** מודיעות ל-device driver להעביר מספר מסוים של בתים בין מאגר שהוגדר על-ידי היישום לבין המאגר בהתקן. לאחר שהבקשה בוצעה, נמצא בה את המספר של הבתים שהועברו בפועל.

הפקודה **Peek** (Nondestructive Read No Wait) מודיעה ל-device driver להעביר למערכת הקבצים העתק של הבית הראשון הנמצא במאגר של ההתקן מבלי להוציא אותו מתוך המאגר. אם המאגר ריק באותו רגע, ה-device driver מחזיר קוד המציין זאת. פקודה זו מאפשרת למערכת הקבצים להסתכל תו אחד קדימה במאגר של ההתקן, מבלי שבקשת **Read** תמתין ב-device driver לקבלת נתונים מההתקן.

הפקודות **Input Status** ו-**Output Status** מבקשות את מצב הק/פ בהתקן התו. מצב הקלט מציין אם יש תווים במאגר של ההתקן, או במלים אחרות, אם יש תווים שאפשר לקרוא אותם. מצב הפלט מציין אם ההתקן תפוס כרגע, כלומר אם תווים נכתבים להתקן.

הפקודות **Input Flush** ו-**Output Flush** מודיעות ל-device driver לבטל את כל הבקשות הממתינות בהתקן, או להפסיק את הביצוע שלהן. השימוש בפקודות אלו נפוץ בעיקר במקרים בהם יש צורך לבטל את הבקשות הממתינות בתור. הן גם משמשות להוצאת נתונים מכל המאגרים (buffers) שב-device driver.

קוד הקטגוריה	תיאור	ה-device driver להתקן התו
01h	התקן אסינכרוני	תקשורת אסינכרונית
04h	מקלדת	מקלדת
05h	מדפסת	מדפסת
07h	עכבר	עכבר
03h	מחוון הציור	מחוון העכבר
0ah	תכנית הפיקוח להתקן תו	כל אחד

תרשים 48. קטגוריות ה-IOCtl עבור device drivers המטפלים בהתקני תו

הפקודות **Device Open** ו-**Device Close** מודיעות ל-device driver שתהליך כלשהו ביישום מתחבר ומתנתק מההתקן. הדבר מאפשר ל-device driver לעקוב אחר השימוש של היישום בהתקן, ולארגן את הגישה להתקן בצורה סידרתית למשל, כדי לאפשר רק לתהליך אחד לבצע ק/פ בו-זמנית. דוגמה אחרת יכולה להיות רישום של מספר השימושים בהתקן (על-ידי הוספה של 1 לכל Open וחיסור של 1 לכל Close). בהתבסס על נתון זה, כאשר מספר זה מגיע לאפס, הוא יוכל להעביר את ההתקן למצב מסוים, או לרוקן את ה-buffers של ההתקן.

הפקודה **Generic IOCtl** נשלחת ל-device driver כתוצאה מבקשת IOCtl שנעשתה על-ידי היישום (הפונקציה DosDevIOCtl, או הפונקציה Int 21h). ה-device driver בודק את הקטגוריה של הפונקציה ואת קוד פונקציית המשנה כדי לקבוע איזה פעולה הוא צריך לבצע. בתרשים 48 תמצא סיכום של הקטגוריות של פקודות IOCtl המתייחסות ל-device driver המטפלים בהתקני תווים.

בתוך כל קטגוריה יש פונקציות משנה המותאמות להתקן מסוים. הקטגוריה של ההתקן האסינכרוני מאפשרת ליישום לטפל במאפייני הקו (כמו למשל, עצירה, זוגיות וסיביות הנתונים), באותות הבקרה של המודם ובבקרת הזרימה (שדר מיד, XON, XOFF) ועוד. קטגוריית המקלדת מאפשרת ליישום לשלוט בפרמטרים של המקלדת הכוללים טבלת תרגום, מצב היסט, ואופן הקלט. קטגוריית המדפסת כוללת פונקציות משנה שמאפשרות ליישום לקבוע את בקרת המסגרת (שורות ותווים באינץ') ולשנות גופנים (פונטים). קטגוריית העכבר כוללת פונקציות משנה המאפשרות ליישום לשלוט במחוון העכבר ולקבל מידע על התקן העכבר עצמו. קטגוריית מחוון הציור קשורה בצורה הדוקה לקטגוריית העכבר ומשמשת בעיקר לקבלת מידע על מחוון העכבר ביחס למסך. קטגוריית תכנית הפיקוח להתקני תו משמשת למשלוח פקודות ייחודיות ל-device driver התומך בתכנית פיקוח להתקנים שלו.

הפקודה **DeInstall** מבקשת מה-device driver להפסיק את התמיכה שלו בהתקן. ה-device driver יכול למלא אחר הבקשה ויכול לסרב לה. אם הוא ממלא אחר הבקשה, הוא חייב לשחרר את כל המשאבים שהוקצו להתקן, כמו זיכרון פיסי ופסקי חומרה. ביצוע הפקודה מתרחש רק בזמן תיחול המערכת ומשמעותה היא ש-device driver חדש שמוטען כעת מבקש להחליף את זה שהוטען קודם. במלים אחרות, ה-device driver רוצה לתמוך בשם ההתקן שנוצר כבר על-ידי ה-device driver הקודם. ה-device driver הישן מתבקש לבצע **DeInstall**

לתמיכה שלו בהתקן שמדובר בו. אם הוא מבצע זאת, אז ה-device driver החדש צריך לבצע **Initialize**. שאם לא כן, הוא יבוטל, כי ה-device driver הישן לא יוותר על הבעלות על ההתקן.

שירותי מערכת ל-DEVICE DRIVERS

רבות מהפעילויות שה-device driver מבצע אינן מכוונות לצורך בקרה על ההתקן בלבד. כפי שתזכור, ה-device drivers משתפים פעולה עם מערכת ההפעלה בטיפול בק/פ של היישום המגיע משתי סביבות ביצוע שונות, במצב אמיתי או במצב מוגן. בנוסף לכך, device drivers משתתפים בתמיכה בביצוע רב-משימתי שמערכת ההפעלה מספקת. מערכת ההפעלה עוזרת ל-device driver לשתף עימה פעולה, על-ידי כך שהיא מספקת לו מספר שירותי מערכת. מערכת ההפעלה מספקת מספר שירותים לפונקציות הנדרשות בד"כ על-ידי device drivers ועל-ידי כך היא מסירה את הצורך לשכפל אותן לכל אחד מהם בנפרד. שירותי מערכת אלה ידועים כשגרות-עזר, או כשירותי DevHlp.

ה-device driver מקבל מחוון למישק DevHlp בחבילת הבקשה של הפקודה Initialize. הוא חייב לשמור את המחוון הזה כדי שיוכל להשתמש בשירותים של המישק. המחוון למישק DevHlp הינו מחוון מיוחד שנמצא בתוקף גם כאשר ה-CPU משנה מצב, ממצב אמיתי למצב מוגן ולהיפך. זאת אומרת שה-device driver יכול תמיד לקרוא למישק DevHlp מבלי להתחשב במצב המעבד. כדי להפעיל שירות DevHlp מסוים, יש לבצע 3 פעולות:

- להציב משתני ביצוע לתוך האוגרים שצוינו על-ידי שירות DevHlp,
- להטעין את קוד הביצוע של השירות לתוך האוגר DL,
- לבצע קריאה רחוקה בעזרת המחוון למישק DevHlp.

ניתן תמיד לקרוא למישק DevHlp, אך הזמינות של השירותים שלו תלויה בנסיבות, או בהקשר שבו הופעל ה-device driver. אנו נדון בהקשרים השונים של פעילויות אלו בפרק זה בסעיף "מרכיבי ה-device driver וההקשרים שלהם".

ניתן לסווג את שירותי ה-DevHlp בקבוצות הבאות:

- ניהול תהליכים.
- ניהול אתתים.
- ניהול תור הבקשות.
- ניהול תור התווים.
- ניהול זיכרון.
- ניהול פסקים.
- שירותי שעון-עצר.
- ניהול תכניות פיקוח להתקני תו.
- ניהול של Advanced BIOS.

כדי לפשט את הדיונים הבאים, תיארנו את שירותי DevHlp בשמות מנמוניים ולא בקוד הביצוע שלהם.

ניהול תהליכים

שירותי DevHlps לניהול תהליכים עוזרים ל-device driver לבצע את הדברים הבאים:

- להשיג מידע הקשור לתהליך.
- לטפל בהפסקות ביצוע של ה-thread.
- לדווח למערכת ההפעלה על אירועים מסוימים של התהליך.
- להכנס לסגמנטים המכילים מידע כללי של המערכת.

שירותי DevHlp בקבוצה זו הם:

- **Block** ו-**Run**
- **Yield** ו-**TCYield**
- **Done**
- **SendEvent**
- **BIOSCritSection**
- **GetDOSVar**

Block ו-**Run** מאפשרים ל-device driver לעצור ולהתחיל בהתאמה את הביצוע של thread. יש לחסום (**Block**) את הביצוע של thread כאשר הוא חייב לחכות שיתרחש אירוע בהתקן לפני שהוא יוכל להשלים את המשימה שצוינה בחבילת הבקשה (שהופקה לפי בקשת היישום). במלים אחרות, הוא משחרר את ה-CPU בזמן שהוא ממתין לפרק זמן שיחלוף, או לפסק בהתקן. כתוצאה מסיום פרק הזמן, או כתוצאה מפסק בהתקן, ה-device driver יוכל להריץ (**Run**) שוב את ה-thread. זאת אומרת, שהוא מאפשר ל-thread לקבל חזרה את ה-CPU כדי שהוא יוכל להמשיך להתבצע.

Yield ו-**TCYield** מאפשרות ל-device driver להפסיק בתנאים מסוימים את הביצוע של ה-thread שלו באופן זמני. **Yield** גורם ל-thread אחד לוותר על ה-CPU ל-thread אחר עם עדיפות ביצוע שווה כשלו או גבוהה ממנה. **TCYield** גורם ל-thread של ה-device driver לוותר על ה-CPU רק ל-thread אחר עם סיווג קריטי מבחינת הביצוע. אם חולף זמן רב (מעל שלוש אלפיות השניה) עד אשר ה-device driver יכול לחסום thread אחר, או לחזור למערכת ההפעלה, הוא חייב לבצע את Yield ו-TCYield במחזוריות.

ה-device driver יכול להשתמש בפקודה **Done** בזמן קבלת פסק, כדי לסמן שהפעולה שנדרשה על-ידי חבילת הבקשה הסתיימה. הוא חייב לעשות זאת רק אם לא נחסם הביצוע של ה-thread בזמן שטיפל בחבילת הבקשה. ה-device driver משתמש ב-**SendEvent** כדי להודיע למערכת ההפעלה על אירועים מסוימים המשפיעים על המערכת כמו לחיצה על המקשים Control+Break, או על צירוף אחר של מקשים המשמש למיתוג בין ה-sessions. כרגע, **SendEvent** מותאם להתקני-קלט אשר בשליטת המשתמש, כמו המקלדת או העכבר.

BIOSCritSection עוזר ל-device driver לתמוך בק/פ להתקן שלו בסביבת DOS. ה-device driver משתמש בו כדי לסמן קטע ביצוע קריטי של ROM BIOS ועל-ידי כך למנוע מהמפעיל לעבור מסביבת ה-DOS. זה חשוב, מאחר והביצוע בסביבת DOS מושעה בכל פעם שהמפעיל יוצא מיישום ה-DOS. אם יישום DOS שולח בקשת ק/פ ל-BIOS, ה-BIOS בתגובה משנה את מצב ההתקן, ויש צורך לאפשר לו לסיים את הפעולה שלו, כדי שההתקן יוכל להיות במצב ידוע לגבי ה-device driver. אפשר להגיד שה-device driver משתמש בפקודה ב-**BIOSCritSection** לסימון כניסה ויציאה מה-BIOS. הוא גם מודיע בעזרתו למערכת ההפעלה, שהמפעיל יכול לעבור מיישום ה-DOS.

GetDOSvar מאפשר ל-device driver להכנס לסגמנט מידע של המערכת ולסגמנט מידע הקשור לתהליך. סגמנט המידע הגלובלי מכיל פרטים שונים, כמו תאריך וזמן וקוד הכונן שממנו הוטענה מערכת ההפעלה. סגמנט המידע המקומי מכיל פרטים כמו מספר הזיהוי של התהליך, עדיפות הביצוע הנוכחית, ומספר ה-session.

ניהול אתתים

שירותי DevHlps אלה מאפשרים ל-device driver להשתמש הן באתתי ה-RAM והן באתתי המערכת כדי לקשר בין רכיבים שלו לבין עצמם וגם בינו לבין יישומים. ה-DevHlps שמאפשרים ניהול אתתים הם: **SemHandle**, **SemRequest**, **SemClear**. אתת מזוהה על-ידי מקש, או על-ידי מקשר לאתת. המקשר לאתת ה-RAM הוא הכתובת שלו (ערך בורר:כתובת יחסית, או סגמנט:כתובת יחסית). את המקשר לאתת המערכת יש לקבל בעזרת **SemHandle**, המעביר מקשר לאתת ברמת היישום. ברגע של-device driver יש מקשר, הוא יכול להשתמש ב-**SemRequest** וב-**SemClear** כדי לטפל הן באתתי ה-RAM והן באתתי המערכת. בעזרת **SemRequest** ניתן לדרוש את האתת ו-**SemClear** משמש לשחרור האתת. בד"כ ה-device driver משתמש באתתי RAM כדי לתאם פעילויות בתוך עצמו, ובאתתי מערכת - כדי להעביר אירועים ליישומים.

ניהול תור הבקשות

שירותי DevHlps אלה מאפשרים ל-device driver לתחזק רשימה מקושרת של חבילות בקשה (request packets). השירותים לניהול תור הבקשות הם:

- **Append**
- **Remove**
- **AddSorted**
- **RemoveSpecific**
- **Alloc** ו-**Free**

device driver, שיכול לטפל בבקשות ק/פ רבות להתקן שלו, עורך מעקב אחר הבקשות לביצוע בעזרת תור עבודה (work queue). DevHlps מאפשרים ניהול פשטני של רשימה מקושרת בעזרת שדה הקישור, אשר נמצא בחבילת הבקשה ומאפשר את שרשור הבקשות. בדרך זו, device driver יכול לתחזק בקלות תור עבודה אחד או יותר לכל התקן.

Append (נקרא גם PushReqPacket) מכניס חבילת בקשה לסוף התור המבוקש. **Remove** (נקרא גם PullReqPacket) מוציא חבילת בקשה מתחילת התור המבוקש. **AddSorted** (או SortReqPacket) מכניס חבילת בקשה לתור המבוקש בסדר המוכתב על-ידי שדה הסקטור הראשון.

הדבר נותן בידי ה-device driver אמצעים פשוטים המאפשרים את ארגון הבקשות. **RemoveSpecific** (או PullParticular) מוציא את חבילת הבקשה המבוקשת מהתור, בלי להתייחס למקומה בתור. **Alloc** ו-**Free** מאפשרים ל-device driver לקבל ולשחרר חבילת בקשה ריקה שאפשר למלא אותה. הוא יכול להשתמש בה לצורך מעקב אחר בקשות ק/פ שהונפקו ל-BIOS על-ידי יישום ב-DOS, או לעקוב אחר שלבים של ק/פ בהתקן רב-שלבי.

ניהול תור התווים

שירותי DevHlps המאפשרים את ניהול תור התווים מאפשרים ל-device driver לתחזק מאגר פשוט של תווים. השירותים לניהול תור התווים:

- **Init**
- **Read**
- **Wrire**
- **Flush**

שירותים אלה מאפשרים ל-device driver לערוך מעקב אחר תווים, בהתבסס על ההנחה שהמידע על תו הוא בגודל של בית אחד. **Init** קובע את המאגר. פעולה זו חייבת להתבצע לפני שמפעילים DevHlps אחרים מקבוצה זו. **Read** ו-**Wrire** מוציאים תו מה-buffer או מוסיפים אליו בהתאמה. **Flush** מאתחל את ה-buffer למצבו הראשוני, כלומר מרוקן אותו.

ניהול זיכרון

שירותי DevHlps לניהול הזיכרון מאפשרים ל-device driver לנהל מיעון בזמן קבלת פסק, או בזמן ביצוע משימה. השירותים לניהול הזיכרון הם:

- **AllocPhys** ו-**FreePhys**
- **PhysToVirt**
- **Lock** ו-**Unlock**
- **VirtToPhys**
- **UnPhysToVirt**
- **PhysToUVirt**
- **VerifyAcess**
- **AllocGDTSelector**
- **PhysToGDTSelector**
- **RealToProt** ו-**ProtToReal**

AllocPhys ו-**FreePhys** מאפשרים ל-device driver להקצות ולשחרר גוש נעול (שלא ניתן להעברה ולהזזה) של זיכרון פיסי. חייב להיות מספיק זיכרון פיסי כדי להכיל את גוש הזיכרון הזה. ה-device driver יכול לבקש גוש זיכרון הגדול מ-64KB והוא יכול גם לסמן, שגוש הזיכרון ימוקם מעל 1MB, או מתחת ל-640KB. המיקום של גוש הזיכרון יכול להיות קריטי לגבי הביצוע, במיוחד אם צריך לפנות אליו בזמן קבלת פסק. מכיון שזיכרון זה הוא זיכרון פיסי לחלוטין, ה-device driver מקבל כתובת פיסית בת 32 סיביות ולא ערך_בורר:כתובת_פיסית או סגמנט:כתובת_פיסית. גוש זה לא ניתן להעברה ולהזזה. כדי להכנס לזיכרון זה, חייבים להפוך את הכתובת הפיסית לכתובת בפועל בעזרת **PhysToVirt**, אשר יוצר כתובת לוגית (ערך_בורר:כתובת_פיסית או סגמנט:כתובת_פיסית) המבוססת על המצב הנוכחי של היע"מ. אם הכתובת נמצאת מעל 1MB והיע"מ נמצאת במצב אמיתי, אז ה-**PhysToVirt** משנה את הסביבה כדי לאפשר ל-device driver להכנס לזיכרון בעזרת כתובת לוגית.

ה-device driver משתמש ב-**Lock** ו-**Unlock** כדי לשנות סגמנט של תהליך, אשר מותר להזזה ולהעברה, למצב קבוע (fix - לא ניתן להעברה ולהזזה) וחזרה, למצב חופשי. כדי להעביר נתונים, ה-device driver צריך לקבוע את הסגמנט של התהליך שהוא מתכוון לגשת אליו באמצעות DMA, או בזמן קבלת פסק.

ה-device driver משתמש ב-VirtToPhys לאחר הנעילה של הסגמנט כדי להפוך את הכתובת הלוגית למספר פיסי בן 32 סיביות. בזמן קבלת הפסק, ה-device driver הופך את המספר הפיסי לכתובת לוגית זמנית בעזרת PhysToVirt. לאחר שהוא סיים את העבודה עם הכתובת הלוגית הזמנית, הוא משתמש ב-UnPhysToVirt. כדי לקבל אפשרות מיעון לזיכרון המסופק על-ידי מתאם (אשר נמצא בד"כ במרחב הכתובות השמור ל-BIOS – 640KB עד 1MB - או מחוץ למרחב הכתובות השמור בד"כ למערכת), ה-device driver יכול להשתמש ב-PhysToUVirt כדי ליצור מתאר LDT לזיכרון זה.

ה-device driver משתמש ב-VerifyAcess כדי לבדוק אם ליישום יש הרשאה להכנס לסגמנט. AllocGDTSelector מופעל כאשר מתבצע תיחול של ה-device driver כדי שיוכל להקצות קבוצה של ערכי בורר לטבלאות ה-GDT לשימושו הפרטי. השימוש ב-PhysToGDTSelector נחוץ כדי לכוון את אחד מערכי הבורר שלו למיקום פיסי בזיכרון, אשר מזוהה ככתובת בת 32 סיביות + אורך. מתאר ה-GDT מתמלא במידע מתאים ונשאר בתוקף עד אשר מתבצעת קריאה נוספת ל-PhysToGDTSelector לאותו ערך בורר. כדי להשתמש בערכי בורר אלה בזמן קבלת פסק, ה-device driver חייב לבדוק את המצב שבו נמצא המעבד. אם המעבד נמצא במצב אמיתי, ה-device driver משתמש בשירות RealToProt כדי לשנות את המצב הזה למצב מוגן. לאחר שהמעבד עבר למצב מוגן, אפשר להשתמש בערכי הבורר האלה כדי להביא ולעבד נתונים. אם ה-device driver צריך לשנות מצב במעבד בזמן קבלת פסק, הוא חייב להעביר אותו למצב אמיתי בעזרת השירות ProtToReal.

ערכי הבורר שנוצרו על-ידי שירותי ה-DevHlps הנ"ל אינם מייצגים סגמנטים רגילים ברמת-היישום. אלה ניתנים להעברה ולהזזה במהלך הפעולה הרגיל של המערכת. סגמנטים או ערכי בורר, שנוצרו על-ידי ה-device driver מנוהלים בצורה שונה מסגמנטים ברמת היישום. בגלל מאפיינים אלה, הם משמשים את ה-device driver בלבד, ואינם ניתנים להעברה. במלים אחרות, ה-device driver משתמש בערכי בורר/סגמנטים אלה לאחסון נתונים.

ניהול פסקים

שירותי DevHlps לניהול פסקים מאפשרים ל-device driver לנהל את פסקי ההתקן:

- SetIRQ ו-UnSetIRQ
- SetSWVector
- EOI

SetIRQ ו-UnSetIRQ מאפשרים ל-device driver לרשום לעבודה או לבטל את הרישום של שיגרה לטיפול בפסק של ההתקן. SetSWVector מאפשר להפעיל שיגרה לטיפול בפסקים של התוכנה בסביבת DOS. EOI שולח את האות "End-Of-Interrupt" (סוף פסק) לבקרי הפסקים המטפלים בפסק המתאים.

שירותי שעון-עצר

שירותי DevHlps לשעון העצר מאפשרים ל-device driver לבצע עיבודים הקשורים בזמן:

- SetTimer
- ResetTimer
- TickCount

ה-device driver משתמש ב-SetTimer כדי לסמן נקודת התחלה לשיגרת הטיפול בשעון העצר, שתופעל לאחר כל תקתוק בשעון. **ResetTimer** מפסיק את פעולתה של שיגרת הטיפול בשעון העצר. **TickCount** משמש לקביעת מספר התקתוקים שיש להמתין לפני שהשיגרה לטיפול בשעון העצר תקרא שוב.

ניהול תכניות פיקוח להתקני תו

שירותי DevHlps מאפשרים ל-device driver להשתתף עם תכניות פיקוח ברמת היישום בעיבוד של זרם הנתונים להתקן. השירותים הם:

- **Create**
- **Register** ו-**DeRegister**
- **Write**
- **Flush**

Create מאתחל רשימת פיקוח להתקן, לפני הרישום של תכנית פיקוח ברמת היישום. **Register** ו-**DeRegister** מוסיפים ומוציאים תכניות פיקוח מרשימת הפיקוח להתקן. **Write** שולח את נתוני התקן לתכניות הפיקוח הנמצאות ברשימה הנ"ל. **Flush** שולח אות-סימון לתכניות הפיקוח כדי להודיע להן למחוק את כל הנתונים ולאתחל את המידע על מצב העבודה הפנימי שלהן.

ניהול של Advanced BIOS

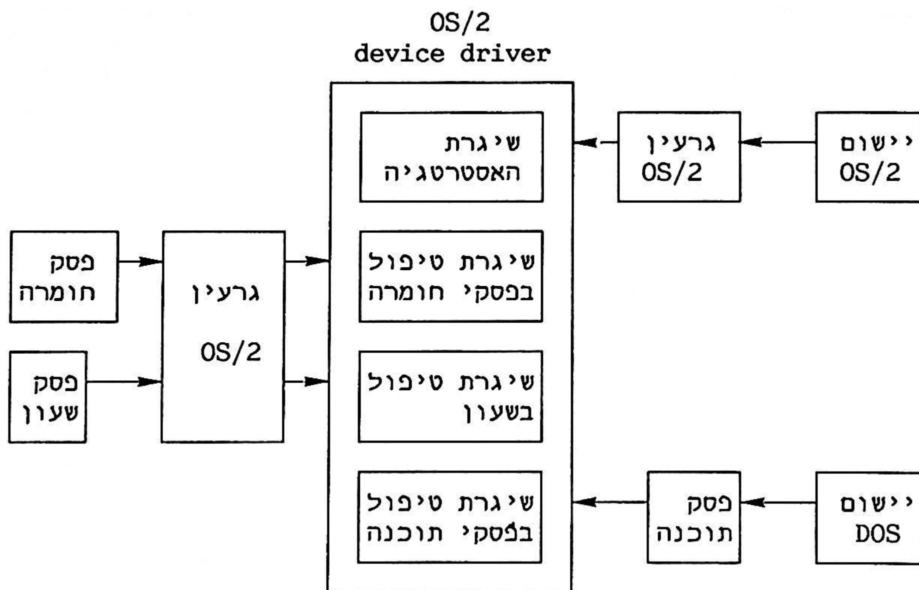
שירותי DevHlps לניהול ה-Advanced BIOS מאפשרים ל-device driver להשתמש בשירותים שלו כאשר הם מסופקים על-ידי החומרה. במקרה זה מדובר ב-PS/2 דגמים 50, 60, 70 ו-80. השירותים בקבוצה זו הם:

- **GetLIDEntry** ו-**FreeLIDEntry**
- **ABIOSCall**
- **ABIOSCommonEntry**

ה-device driver משתמש ב-**GetLIDEntry** ו-**FreeLIDEntry** כדי לקבל ולשחרר מספר זיהוי לוגי להתקן הנתמך על-ידי ה-Advanced BIOS. יש צורך להשיג מספר זיהוי לוגי לפני שנעשית קריאה לפונקציה של ה-Advanced BIOS. **ABIOSCall** ו-**ABIOSCommonEntry** קובעות את המסגרת הדרושה למחסנית וקוראות לפונקציה של Advanced BIOS.

מרכיבי ה-DEVICE DRIVER

ה-device driver ב-OS/2 מורכב ממרכיב אחד או יותר המשתפים פעולה בניהול הק/פ להתקן. שני המרכיבים העיקריים הם: שיגרת האסטרטגיה (Strategy Routine), שתמיד נמצאת ב-device driver, ושיגרת הטיפול בפסקי חומרה (Hardware Interrupt Handler), אשר נדרשת אם ההתקן יכול לספק פסק חומרה. שני מרכיבים אפשריים נוספים הם: שיגרת הטיפול בשעון העצר (Timer Handler) ושיגרת הטיפול בפסקי תוכנה (Software Interrupt Handler). תרשים 49 מראה את היחסים בין ה-device driver לשאר חלקי מערכת ההפעלה.



תרשים 49. מרכיבי device driver ב-OS/2

שיגרת האסטרטגיה

שיגרת האסטרטגיה היא המרכיב העיקרי של ה-device driver. היא מטפלת בבקשות הק/פ המופקות על-ידי היישום, ולמעשה - בכל חבילות הבקשה שמערכת ההפעלה שולחת ל-device driver. מערכת ההפעלה קובעת את נקודת הכניסה לשיגרת האסטרטגיה בשדה המוגדר בכותרת ההתקן, ובכך היא יכולה לקרוא ל-device driver עם חבילת הבקשה של הפקודה INIT. פקודה זו מודיעה ל-device driver לתחל את עצמו (כאשר הוא מותקן) בעת התיחול הכללי של המערכת.

מערכת ההפעלה קוראת לשיגרת האסטרטגיה מטעם היישום, ובכך היא מאפשרת לה לקבל ק/פ מיישום DOS הרץ בסביבת DOS ומיישומי OS/2 רבים הרצים בסביבת OS/2. זאת אומרת ששיגרת האסטרטגיה יכולה לרוץ הן במצב אמיתי והן במצב מוגן (ולכן היא דו-מצבית). האופי הדו-מצבי של ה-device driver משפיע על האופן שבה פועלת שיגרת האסטרטגיה. עיין בבקשה בסעיף הדן בשיקולים של תפעול דו-מצבי.

בעת עיבוד חבילת הבקשה, שיגרת האסטרטגיה מתבצעת בזמן-משימה. כלומר, השיגרה מתבצעת תחת ה-thread של התהליך שביצע את בקשת הק/פ ובתוך המחסנית שלו. מכיון שגרעין מערכת ההפעלה קורא לשיגרת האסטרטגיה, אפשר להגיד שהיא מתבצעת ב"מצב גרעין" (kernel mode). המאפיין החשוב ביותר של ריצה במצב גרעין הוא, שמנתב המשימות (task switch) אינו יכול להפסיק את הביצוע של ה-thread של שיגרת האסטרטגיה. במלים אחרות, ה-thread של שיגרת האסטרטגיה לא יפסיק להתבצע עד אשר הוא ייחסם מרצונו, יוותר ל-thread אחר, או ינסה לגשת לסגמנט שאינו נמצא בזיכרון. מצב השגיאה "לא נוכח" במעבד יאלץ את מערכת ההפעלה להחזיר את הסגמנט לזיכרון המערכת, ובאותו זמן להריץ thread אחר. הדבר נותן בידי

ה-device driver את החופש להחליט מתי הוא יוותר על היע"מ. הויתור על היע"מ חשוב, מאחר והוא יכול לשנות את המצב של ההתקן. כל שינוי מצב חייב להיות לזמן קצר ביותר, או להתבצע עד הסוף כדי להבטיח שההתקן ישאר במצב ידוע. פסק בחומרה יכול להפריע לביצוע של שיגרת האסטרטגיה. לכן, אם לשיגרת האסטרטגיה ולשיגרת הטיפול בפסקי חומרה (או בפסקי שעון העצר) יש גישה משותפת אל מבני נתונים, היא חייבת לתאם את הפעולות שלה עם המרכיבים המופעלים בזמן קבלת פסק.

שיגרת האסטרטגיה נכתבת כמודל של FAR CALL (קריאה רחוקה), FAR RETURN (החזרה רחוקה). מערכת ההפעלה תבצע קריאה רחוקה לשיגרת האסטרטגיה, אשר חייבת לבצע החזרה רחוקה בגמר העבודה. לשיגרה זו יש גישה אוטומטית לסגמנט הנתונים (באוגר ה-DS) של ה-device driver ולמחסנית (באוגר ה-SS), בלי קשר למצב הנוכחי של היע"מ. היא אינה צריכה לשמור ולשחזר את התוכן של האוגרים, כי מערכת ההפעלה עושה זאת עבורה.

הדוגמה הבאה מתארת את הפעילויות של שיגרת האסטרטגיה.

1. מערכת ההפעלה קוראת לשיגרת האסטרטגיה עם מחוון (ES:BX) לחבילת בקשה.
   השיגרה מתבצעת כ-thread במסגרת התהליך שהפיק את קריאת המערכת, והיא משתמשת במחסנית שסופקה על-ידי התהליך הקורא. לשיגרת האסטרטגיה יש אפשרות מיעון לסגמנט הנתונים של ה-device driver (באוגר DS). בכל פעם שהיא נמצאת במצב מוגן, היא מתבצעת ברמת הרשאה 0, הזהה לזו של גרעין מערכת ההפעלה. רמה זו מקבלת בצורה אוטומטית הרשאת ק/פ (IOPL).
2. שיגרת האסטרטגיה מוודאת שחבילת הבקשה בתוקף.
3. - אם ניתן לטפל בפונקציה המבוקשת מיד (כמו בדיקת מצב), שיגרת האסטרטגיה תבצע את הפעולה.
   - אם הפונקציה המבוקשת חייבת להשלח להתקן, השיגרה תוסיף את חבילת הבקשה לתור הממתינים כאשר ההתקן תפוס, או תשלח את חבילת הבקשה להתקן אם הוא פנוי.
4. אם הפונקציה הרצויה הושלמה, שיגרת האסטרטגיה מציבה הודעה בחבילת הבקשה ושולחת אותה לגרעין מערכת ההפעלה.
5. אם הפונקציה הרצויה עדיין ממתינה, שיגרת האסטרטגיה תמתין על-ידי חסימת הביצוע של ה-thread.
6. לאחר שחסמה את ה-thread, שיגרת האסטרטגיה יכולה לשחרר את היע"מ. מערכת ההפעלה תנתב אל המעבד thread אחר, שיוכל לבצע קריאת מערכת. השיגרה תוכל להקרא שוב בנקודת הכניסה שלה.

שיגרת הטיפול בפסקי-חומרה

שיגרת הטיפול בפסקי-חומרה היא רכיב חשוב ביותר של ה-device driver, מכיון שהינה הרכיב **היחיד** במערכת שרשאי לטפל בפסקי החומרה. לכן, שיגרת הטיפול בפסקי-חומרה נדרשת רק בהתקנים המפיקים פסקים. ה-device driver חייב לרשום את נקודת הכניסה לשיגרה זו בעזרת שירות ה-SetIRQ DevHlp. מכאן ואילך, מערכת ההפעלה תקרא לשיגרת הטיפול בפסקי-חומרה בכל פעם שיתרחש פסק בחומרה.

התקן פועל בצורה אסינכרונית לשאר חלקי המערכת, ולכן הוא יכול להפיק פסק בלי קשר ליישום שמתבצע, ובלי קשר למצב שבו נמצא ה-CPU. לדוגמה,

בכל פעם שיישום DOS נמצא בחזית, ניתן לבצע ק/פ מיישום OS/2 אחר, אשר רץ ברקע. לגבי תהליך הנמצא במצב מוגן, ההתקן מסמן את סיום הק/פ על-ידי כך שהוא גורם לפסק במצב אמיתי. במלים אחרות, ביצוע ק/פ מיישום DOS הרץ בחזית, יושלם במשך תקופת הביצוע של יישום OS/2 שברקע. לגבי תהליך הנמצא במצב אמיתי, ההתקן מסמן את סיום הק/פ על-ידי כך שהוא גורם לפסק במצב מוגן. מאחר ופסקי החומרה יכולים להתרחש הן במצב מוגן והן במצב אמיתי, שיגרה זו חייבת להתבצע בהתאם, ולכן היא דו-מצבית.

שיגרת הטיפול בפסקי-חומרה יכולה להתבצע בזמן קבלת פסק בלבד. זאת אומרת, ששיגרה זו מתבצעת תחת thread-זמן-פסק, אשר שונה מ-thread-זמן-משימה בכך שהוא אינו שייך לתהליך של יישום כלשהו. השיגרה מתבצעת במחסנית זמן-פסק מיוחדת, שסופקה על-ידי מערכת ההפעלה. המונח "מצב פסק" מתאר את המשמעות של הביצוע. המאפיין העיקרי של ריצה במצב פסק הוא שלביצוע בזמן פסק יש עדיפות גבוהה יותר מאשר לביצוע בזמן משימה ולכן, מנתב המשימות (task switch) אינו יכול להפסיק ביצוע בזמן-פסק. זאת אומרת, שלצורך השגת רמת ביצועים גבוהה יותר, חשוב ביותר לצמצם למינימום את הזמן המבוזבז על-ידי שיגרת הטיפול בפסקי-חומרה. ההשפעה של פעולות בזמן-פסק על פעולות בזמן-משימה היא מצטברת. בזמן שפסקי-חומרה עם עדיפות ביצוע גבוהה מתפרצים לביצוע של thread-זמן-פסק, הוא אינו יכול לשרת פסקי חומרה עם עדיפות ביצוע נמוכה יותר הנכנסים למערכת. לכן חשוב, שהזמן הכולל המבוזבז על-ידי שיגרת הטיפול בפסקי-חומרה על פעילות בזמן-פסק, יהיה נמוך ככל האפשר. בדומה לשיגרת האיסטרטגיה, שיגרת הטיפול בפסקי-חומרה מחליטה מתי לוותר על היע"מ, כלומר מתי להפיק את הסימן "סוף פסק" (End-Of-Interrupt) ולהעביר אותו לבקר הפסקים, ומתי לצאת מהשיגרה. הדבר מאפשר לשיגרת הטיפול בפסקי-חומרה לשמור על כך, שלא יהיו שינויים בלתי צפויים בהתקן.

שיגרת הטיפול בפסקי-חומרה מקודדת כמודל FAR CALL (קריאה רחוקה), FAR RETURN (החזרה רחוקה). מערכת ההפעלה תבצע קריאה רחוקה לשיגרת הטיפול בפסקי-חומרה, אשר חייבת לבצע החזרה רחוקה בגמר העבודה. לשיגרה יש גישה אוטומטית לסגמנט הנתונים (באוגר ה-DS) של ה-device driver ולמחסנית (באוגר ה-SS), בלי קשר למצב הנוכחי של היע"מ. שיגרה זו אינה צריכה לשמור ולשחזר את התוכן של האוגרים, כי מערכת ההפעלה דואגת לתוכן של האוגרים השייכים ל-thread-זמן-פסק.

הדוגמה הבאה מתארת את הפעילויות של שיגרת הטיפול בפסקי-חומרה.

1. מערכת ההפעלה קוראת לשיגרת הטיפול בפסקי-חומרה כאשר מתרחש פסק בהתקן. השיגרה מתבצעת כ-thread מיוחד שאינו שייך לתהליך יישומי כלשהו, ומשתמשת במחסנית שסופקה על-ידי מערכת ההפעלה. לשיגרת הטיפול בפסקי-חומרה יש אפשרות מיעון לסגמנט הנתונים של ה-device driver (באוגר DS). בכל פעם שהשיגרה נמצאת במצב מוגן, היא מתבצעת ברמת הרשאה 0, הזהה לזו של גרעין מערכת ההפעלה. רמה זו מקבלת בצורה אוטומטית הרשאת ק/פ (IOPL).
2. שיגרת הטיפול בפסקי-חומרה מוודאת שההתקן גרם לפסק תקף.
שתי סיבות יכולות לגרום להתקן להפיק פסק: הראשונה - בקשה שנשלחה אליו והשניה - כתוצאה מאירוע שאינו קשור לבקשה. בכל מקרה, שיגרת הטיפול בפסקי חומרה חייבת לאתחל את מצב הפסק בהתקן.

3. אם הפסק הוא תוצאה של פקודה קודמת שנשלחה להתקן, שיגרת הטיפול בפסקי-חומרה מוודאת שהפונקציה המבוקשת תושלם.
   - אם הפונקציה המבוקשת הושלמה, שיגרת הטיפול בפסקי-חומרה תציב את מצב הבקשה בחבילת הבקשה ותריץ את ה-thread הממתין לשיגרת האסטרטגיה.
   - אם הפונקציה המבוקשת עדיין ממתינה (לדוגמה, באמצע ביצוע של פעולה רב-שלבית), שיגרת הטיפול בפסקי-חומרה מתחילה את השלב הבא של הפעולה. כאשר הסתיים השלב האחרון השיגרה יכולה לעורר את ה-thread של שיגרת האסטרטגיה.
4. לאחר שבוצעה הפונקציה הרשומה בחבילת הבקשה, יכולה שיגרת הטיפול בפסקי החומרה לשלוח את הבקשה הבאה להתקן. לאחר מכן היא יכולה לגשת לתור הבקשות הממתינות, ולטפל בבקשה הבאה הממתינה לביצוע.
5. לאחר ששיגרת הטיפול בפסקי-חומרה סיימה את פעולותיה לגבי התרחשות זו של פסק בחומרה, היא חוזרת אל גרעין מערכת ההפעלה שקרא לה.

השיגרה תוכל להקרא שוב בנקודת הכניסה שלה בכל זמן לאחר שהודיעה לבקר הפסקים על מצב של "סוף פסק" (EOI). מצב סוף פסק מודיע לבקר הפסקים שניתן כעת לשרת כל פסק הממתין אצלו כדי להשלח ליע"מ. מכאן אפשר להבין, שפסק נוסף יכול להמתין בבקר הפסקים בציפיה לשירות של שיגרת הטיפול בפסקי החומרה. זאת אומרת, שבפרק הזמן בין הודעת ה-EOI לבין החזרה למערכת ההפעלה, ניתן לקרוא לשיגרה בנקודת הכניסה שלה. פעולה זו ידועה בשם קינון-פסקים (interrupt nesting).

שיגרת הטיפול בשעון העצר

שיגרת הטיפול בשעון העצר הינה מרכיב אופציונלי של ה-device driver. היא משמשת בעיקר לניהול פעולות התלויות בתזמון, ואפשר לדמות אותה לשעון העצר BIOS Int 1Ch בסביבת ה-DOS. אולם, שיגרה זו מופעלת בזמן אמיתי על-ידי שעון CMOS ולא על-ידי שעון המערכת. מכיון ששיגרת הטיפול בשעון העצר מופעלת על-ידי פסק בחומרה, ה-device driver יכול להשתמש בה כדי לדעת מתי לחקור התקנים שאינם מבוקרי-פסקים. ה-device driver חייב לרשום את נקודת הכניסה לשיגרה זו בעזרת שירות DevHlp TickCount, שבו צריך לציין את מספר ה"תקתוקים" שיעברו בשעון לפני ששיגרה זו תופעל. אפשר לרשום את נקודת הכניסה גם בעזרת שירות DevHlp אחר, SetTimer, שבו מרווח הזמן שווה לתקתוק אחד. לאחר שנרשמת נקודת הכניסה, מערכת ההפעלה יכולה לקרוא לשיגרה זו בכל פעם שחולף פרק הזמן הדרוש.

שעון ה-CMOS עובד בצורה אסינכרונית לשאר המערכת, ולכן הוא יפיק פסקי חומרה (במקרה הזה, תקתוקי שעון) בלי קשר למצב היע"מ. פסקים בהתקן יכולים להתרחש הן במצב אמיתי והן במצב מוגן, ולכן שיגרת הטיפול בשעון העצר עובדת בשני המצבים, כלומר היא דו-מצבית. האופי הדו-מצבי של ה-device driver ב-OS/2 משפיע על הדרך שבה שיגרה זו פועלת. בסעיף "שיקולים בעבודה בשני מצבים" תמצא דיון על ההשלכות של עבודה בשני מצבים.

בדומה לשיגרת הטיפול בפסקי חומרה, שיגרה זו מתבצעת בזמן-פסק, תחת thread-זמן-פסק במצב פסק. לכן, זמן העיבוד בתוך שיגרה זו חייב להיות מינימלי, כי יש לקחת בחשבון שיכולות לפעול במערכת מספר שגרות כאלו.

שלא כמו שיגרת הטיפול בפסקי חומרה, שיגרה זו אינה מפיקה הודעת "סוף פסק", כי מערכת ההפעלה מנהלת את התקן השעון. פרט לזאת, כל שאר שיקולי הביצוע לגבי שיגרה זו דומים לאלה של שיגרת הטיפול בפסקי חומרה.

שיגרת הטיפול בשעון העצר מקודדת כמודל FAR CALL, FAR RETURN. מערכת ההפעלה תבצע קריאה רחוקה לשיגרת הטיפול בשעון העצר, אשר חייבת לבצע החזרה רחוקה בגמר העבודה. לשיגרת הטיפול בשעון העצר יש גישה אוטומטית לסגמנט הנתונים (באוגר ה-DS) של ה-device driver ולמחסנית (באוגר ה-SS), בלי קשר למצב הנוכחי של היע"מ. שיגרת הטיפול בשעון העצר אינה צריכה לשמור ולשחזר את התוכן של האוגרים, כי מערכת ההפעלה דואגת לתוכן של האוגרים השייכים ל-thread-זמן-פסק.

הדוגמה הבאה מתארת את הפעילויות של שיגרת הטיפול בשעון העצר.

1. מערכת ההפעלה קוראת לשיגרת הטיפול בשעון העצר כאשר מתרחש פסק בהתקן.
השיגרה הזו מתבצעת כ-thread מיוחד שאינו שייך לתהליך של יישום כלשהו, ומשתמשת במחסנית שסופקה על-ידי מערכת ההפעלה. לשיגרת הטיפול בשעון העצר יש אפשרות מיעון לסגמנט הנתונים של ה-device driver (באוגר DS). בכל פעם שהיא נמצאת במצב מוגן, היא מתבצעת ברמת הרשאה 0, הזהה לזו של גרעין מערכת ההפעלה.
2. שיגרת הטיפול בשעון העצר יכולה:
   - לפעול בפרקי זמן קצובים.
   - לטפל בק/פ של התקנים שאינם מבוקרי-פסקים.

   אם יש צורך, שיגרה זו יכולה לעורר את ה-thread ששיגרת האסטרטגיה חסמה.
3. כאשר שיגרת הטיפול בשעון העצר מסיימת את פעולותיה לגבי התרחשות זו של פסק הזמן, היא חוזרת לגרעין מערכת ההפעלה שקרא לה. שיגרת הטיפול בשעון העצר אינה מפיקה הודעת "סוף פסק" (EOI) לבקר פסקי החומרה, כי הבעלות על התקן השעון נמצאת בידי ה-device driver של השעון.

שיגרת הטיפול בפסקי תוכנה

שיגרת הטיפול בפסקי תוכנה הינה מרכיב אופציונלי של ה-device driver והיא משמשת רק לתמיכה בסביבת DOS. תפקידה ליירט את פסקי התוכנה שהופקו על-ידי יישום DOS. ה-device driver חייב לרשום את נקודת הכניסה לשיגרה זו בעזרת שירות SetSWVector (הידוע גם כ-SetROMVector) של ה-DevHlp. לאחר מכן, בכל פעם שסביבת הביצוע של DOS נמצאת בחזית, יכול יישום ה-DOS להפיק פסק תוכנה, אשר יפעיל בצורה ישירה את שיגרת הטיפול בפסקי התוכנה.

פסקי תוכנה יכולים להתרחש אך ורק בסביבת DOS, ולכן שיגרה זו מתבצעת רק במצב אמיתי. למעשה, שיגרה זו הינה היחידה ב-device driver שאינה דו-מצבית.

שיגרה זו נקראת ישירות על-ידי פסק התוכנה ולכן היא מתבצעת בזמן-משימה כהרחבה של היישום. הקשר זה של הביצוע נקרא "מצב משתמש", אשר המאפיין העיקרי שלו הוא בכך שמנתב המשימות יכול להפסיק את הביצוע של משימה זו. זאת אומרת, ששיגרת הטיפול בפסקי תוכנה נחשבת כחלק מיישום ה-DOS, והיא

יכולה להפסיד את היע"מ לתהליכי OS/2 הנמצאים ברקע. יש צורך להתחשב בתכונה זו של "מצב משתמש", אם שיגרת הטיפול בפסקי תוכנה חייבת לגשת למבני נתונים המשמשים מרכיבים אחרים של ה-device driver. לדוגמה, שיגרה זו יכולה להתחיל לעדכן מבנה נתונים ואז "לאבד" את היע"מ לתהליך הנמצא ברקע. תהליך זה יכול לבצע בקשת ק/פ, שמפעילה את שיגרת האסטרטגיה, אשר בתורה תעדכן את אותו מבנה נתונים. הביצוע של שיגרת הטיפול בפסקי תוכנה יכול להיות מופרע על-ידי פסקי חומרה בנוסף לניתוב המשימות. זאת אומרת, ששיגרת הטיפול בפסקי התוכנה חייבת לקחת בחשבון את הסיכונים כאשר היא משתמשת במבני נתונים המשמשים מרכיבים אחרים של ה-device driver הפועלים בזמן-פסק (השגרות לטיפול בפסקי החורה ובשעון העצר).

שימוש אופייני של ה-device driver בשיגרה זו הוא יירוט של פסק תוכנה המופק על-ידי ה-ROM BIOS, אשר מספק ליישום DOS גישה להתקן המנוהל על-ידי device driver. אחת הסיבות שבגללן device driver רוצה לדעת מה ה-BIOS מנסה לעשות, היא הצורך להבטיח שההתקן ישאר במצב ידוע. במקרה זה, ה-device driver מטפל בבקשות ק/פ שמופקות על-ידי תהליכי OS/2 הנמצאים ברקע. לכן הוא חייב לתאם את הגישה להתקן בינו לבין ה-BIOS, ולהבטיח שהגישה להתקן תהיה סידרתית. סיבה אחרת היא להגן על שירות ה-BIOS מפני הפסקה לפני שהוא השלים את פעולתו, וגם כאן, ה-device driver מעוניין להשאיר את ההתקן במצב ידוע. במצב זה, הבעיה היא, שהמפעיל יכול לעבור בכל רגע מיישום ה-DOS הנמצא בחזית ולהעביר לשם יישום OS/2 שהיה ברקע. כתוצאה מכך יושעה הביצוע של יישום DOS. אם יישום DOS הפעיל פסק תוכנה והמפעיל עבר ליישום אחר, שירות BIOS לא יושלם. דבר זה יכול להיות חמור מאוד אם שירות ה-BIOS היה באמצע פעולה של שינוי מצב בהתקן. כדי למנוע זאת, שיגרת הטיפול בפסקי תוכנה מסמנת בעזרת שירות ה-DevHlp, BIOSCritSection, ששירות BIOS, נמצא בשלב קריטי של ביצוע ועל-ידי כך היא מונעת מהמשתמש מלעבור באופן זמני מיישום DOS.

שיגרת הטיפול בפסקי תוכנה אינה נקראת על-ידי מערכת ההפעלה אלא על-ידי פסק תוכנה. לכן, היא כתובה כמודל INTERRUPT (פסק), Interrupt RETurn - IRET (החזרת פסק). פסק בתוכנה יפעיל את שיגרת הטיפול בפסקי תוכנה בעזרת הצבה מסוימת במחסנית של יישום DOS שמדובר בו. השיגרה חייבת להחזיר פסק לאחר שסיימה את פעולתה. בנקודת הכניסה, חייבת שיגרה זו לקבל הרשאת גישה לסגמנט הנתונים של ה-device driver. בנוסף לכך, עליה לשמור ולשחזר את התוכן של האוגרים בשמו של יישום DOS שקרא לה.

הדוגמה הבאה מתארת את הפעולות של שיגרת הטיפול בפסקי תוכנה.

1. שיגרת הטיפול בפסקי תוכנה נקראת על-ידי יישום DOS, אשר הפעיל פסק בתוכנה. השיגרה מתבצעת כשלוחה של היישום ומשתמשת במחסנית של היישום שקרא לה. שיגרה זה חייבת להסדיר אפשרות מיעון לסגמנט הנתונים של ה-device driver על-ידי כך שהיא מציבה בתוך אוגר הסגמנט DS ערך שנשמר קודם. שיגרת הטיפול בפסקי תוכנה מתבצעת במצב אמיתי.
2. שיגרת הטיפול בפסקי תוכנה מוודאת שהפונקציה המבוקשת ניתנת לביצוע.
3. אם התקבלה הרשאה לבצע את הפונקציה הרצויה, יכולים להיות שני מצבים:

- ההתקן פנוי: שיגרה זו תפעיל BIOSCritSection כדי שתוכל להכנס לשלב קריטי של הביצוע, ולאחר מכן היא תקרא ל-BIOS כדי שיבצע את הפונקציה. לאחר שה-BIOS סיים את פעולתו, היא מפעילה שוב BIOSCritSection כדי לצאת מהשלב הקריטי של הביצוע.
- ההתקן תפוס: שיגרה זו תקבע אות-סימון שיצביע על כך שק/פ של ה-BIOS נמצא בתור להתקן ותמתין באתת RAM בעזרת שירות SemRequest של ה-DevHlp. כאשר האתת פנוי (לאחר ששיגרת האסטרטגיה או ששיגרת הטיפול בפסקי חומרה הפעילו SemClear), ימשיך הק/פ של ה-BIOS להתבצע כפי שצוין לעיל.

4. שיגרת הטיפול בפסקי תוכנה חוזרת ליישום ה-DOS על-ידי החזרת פסק (IRET).

בכל זמן שהוא בעת הביצוע, יכולה להיות הפסקה בפעולת השיגרה לטיפול בפסקי תוכנה, לשם קבלת פלחי זמן (timeslice) עבור יישומים הפועלים ברקע במצב מוגן. הביצוע של השיגרה יושעה אם המשתמש העביר את סביבת ה-DOS לרקע. הביצוע לא יושעה אם השיגרה הפעילה BIOSCritSection כדי ששלב קריטי יבצוע ללא הפרעה. לגבי הפעלה חוזרת, שיגרת הטיפול בפסקי תוכנה לא תופעל בנקודת הכניסה עד לאחר שהיא תחזור ליישום ה-DOS.

שיקולי עבודה בשני מצבים

ה-device drivers ב-OS/2 הם **דו-מצביים** (bimodal). שלושה מתוך ארבעת המרכיבים ב-device driver מתבצעים הן במצב אמיתי והן במצב מוגן. היכולת להתבצע בלי קשר למצב היע"מ נקראת עבודה בשני מצבים, או עבודה מעורבת. רק לגרעין ה-OS/2 ול-device drivers יש את היכולת הזו. יש לציין שהדבר שונה מ"יישום משפחה" (Family Application) הנכתב לממשק לתכנות יישומים (API) ומקושר לספריות משפחה מיוחדות, שמאפשרות ליישום המשפחה להתבצע הן ב-OS/2 והן ב-DOS. יישום משפחה מתבצע רק במצב היע"מ שבו הוא הוטען.

ביצוע במצב אמיתי פירושו, שהמרכיבים הדו-מצביים של ה-device driver חייבים להיות מקודדים בהתאם לכללים מסוימים שמכתיבה הארכיטקטורה של OS/2. הכללים הם:

- לא לעשות חישובי סגמנטים בתוך אוגרי הסגמנט.
- לא להיות תלויים בסגמנטים עם אפשרות גלילה (wraparound).
- לא לגשת לכתובות מעבר לסוף הסגמנט.
- להציב ערכי בורר נכונים באוגרי הסגמנט.
- לא לכתוב לתוך סגמנט פקודה.
- לא להציב פקודות ונתונים באותו סגמנט.
- לא להיות תלויים במהירות הביצוע של הפקודה.

לאחר שמלאו אחר כללים אלה, יוכלו הרכיבים הדו-מצביים ב-device driver להתבצע במצב אמיתי. הדבר אפשרי, כי לא קיימות הגבלות על ביצוע פקודות במצב אמיתי.

המשוכה העיקרית העומדת בפני הרכיבים האלה כאשר הם מתבצעים במצב אמיתי, היא המיעון לזיכרון הנמצא מעל 1MB. כפי שבוודאי תזכור, מיעון זיכרון

במצב אמיתי משתמש רק ב-20 סיביות ולכן, הכתובת הגבוהה ביותר שניתן להגיע אליה היא FFFFFh, או במונחים של סגמנט:כתובת_יחסית נכתוב F000:FFFF (שמתורגם ל-1MB). הרכיבים הדו-מצביים ב-device driver חייבים לנקוט בפעולות מסוימות, אם הם רוצים להכנס במצב אמיתי לזיכרון הנמצא מעל 1MB. שלבים אלה מתוארים להלן:

1. לאחר ששיגרת האסטרטגיה ב-device driver קיבלה חבילת בקשה המכילה כתובת של מאגר הנמצא בסגמנט הנתונים של היישום, היא מוודאת שהמאגר נעול בזיכרון (לא ניתן להזזה ולהעברה). כתובות הנמצאות בחבילות הבקשה של READ ו-WRITE ננעלות על-ידי מערכת הקבצים, ולכן ה-device driver אינו צריך לנעול אותן. אולם, כתובות המועברות עם חבילת הבקשה של IOCtl אינן נעולות ויש צורך לנעול אותן. שיגרת האסטרטגיה משתמשת בשירות Lock של ה-DevHlp. כדי לנעול כתובת בזיכרון (לא ניתן לנעול זיכרון בזמן-פסק ולכן רק שיגרת האסטרטגיה יכולה לנעול כתובות בזיכרון).
2. לאחר ששיגרת האסטרטגיה מצאה שהזיכרון נעול, היא חייבת להפוך את הכתובת הלוגית, כלומר ערך_בורר:כתובת_יחסית או סגמנט:כתובת_יחסית, למספר בן 32 סיביות. היא עושה זאת בעזרת שירות VirtToPhys של ה-DevHlp. בגמר ביצוע פעולה זו שומרת שיגרת האסטרטגיה את המספר בן 32 הסיביות. הכתובות הנמצאות בחבילות הבקשה של READ ו-WRITE מיוצגות כבר כמספרים בני 32 סיביות. על-כן יש צורך להסב רק כתובות שה-device driver צריך לנעול.
3. כאשר ה-device driver צריך להכנס לזיכרון, הוא מסב את המספר בן 32 הסיביות לכתובת לוגית. לצורך זה הוא משתמש בשירות ה-DevHlp, PhysToVirt, שמחזיר את המבנה של ערך_בורר:כתובת_יחסית או סגמנט:כתובת_יחסית לגבי המקום בזיכרון. בנוסף לכך, אם היעד נמצא מעל 1MB והיע"מ נמצאת במצב אמיתי, ה-PhysToVirt משנה את מצב היע"מ, כך שה-device driver יוכל להכנס לזיכרון.
4. כאשר ה-device driver יוצא מהזיכרון, הוא חייב להפעיל את שירות ה-DevHlp, UnPhysToVirt. שירות זה מחזיר את מצב היע"מ לזה שהיה לפני שבוצעה הקריאה ל-PhysToVirt.

בעזרת שלבים אלה, ה-device driver אינו צריך לזהות את מצב היע"מ, וניתן לתכנן אותו כך שהוא לא יתחשב במצב זה.

שיקולי תפעול

הפעילויות בזמן משימה של ה-device driver (שיגרת האסטרטגיה ושיגרת הטיפול בפסקי תוכנה, למשל) נעשות במסגרת התהליך שקרא להן, אך הן מושפעות בצורה שונה מיתר חלקי המערכת המשתתפים בביצוע רב משימות (multitasking). מאידך, הפעילויות בזמן-פסק הן מיוחדות, כי הן מתבצעות בצורה אסינכרונית לשאר המערכת. לאור זאת יש ל-device driver מספר מאפיינים ייחודיים, המשפיעים על הפעילות ההדדית של המרכיבים השונים ב-device driver. המאפיינים הם:

- מיעון לזיכרון.
- סינכרון.
- קינון של פסקים.
- ביצועי מערכת.

מיעון לזיכרון

מיעון לסגמנטים של היישום בזמן פסק חייב להתחשב לא רק בעבודה בשני מצבים, אלא גם בהקשר שבו נמצא היישום. אפילו אם נקבע שהמערכת תעבוד רק במצב מוגן, לא פתרנו את הבעיה. הסיבה לכך היא שביצוע רב-משימתי של תהליכים פירושו שהמרכיבים הפועלים בזמן-פסק יתבצעו במרחב כתובות השונה מזה שממנו נשלחה בקשת הק/פ. זאת אומרת, שהמרכיבים הפועלים בזמן-פסק (שגרות הטיפול בפסקי חומרה ובשעון העצר) חייבים להשתמש בשירותי DevHlp לניהול הזיכרון, כדי להבטיח גישה בזמן-פסק לסגמנטים של היישום. המרכיב הפועל בזמן-משימה, שיגרת האסטרטגיה, אחראית להפיכת כל כתובת לוגית המתקבלת מתהליכי היישום לכתובת פיסית ולאחסון של התוצאה כמספר בן 32 סיביות. המרכיבים הפועלים בזמן-פסק יכולים לאחזר את המספר בן 32 הסיביות, להפוך אותו באופן זמני לכתובת לוגית, ולהזיז נתונים פנימה והחוצה מתוך הסגמנט של היישום.

יש צורך בשיקול מיוחד עבור מיעון לזיכרון פיסי הן לגבי זמן-פסק והן לגבי זמן-משימה. על-ידי שימוש בשירות AllocPhys של DevHlp, שיגרת האסטרטגיה מקצה גוש של זיכרון פיסי מתוך זיכרון המערכת ומקבלת מספר בן 32 סיביות. גוש פיסי זה אינו סגמנט שניתן לטפל בו בצורה ישירה, אלא בלוק נעול של הזיכרון, שאינו ניתן להעברה ולהזזה. זהו המנגנון היחיד העומד לרשות ה-device driver כדי להשיג יותר זיכרון ממה שיש לו בסגמנטים שלו. כדי שמרכיבי ה-device driver יוכלו להתייחס לזיכרון זה, הם חייבים להשתמש בשירות ה-DevHlp, PhysToVirt, כדי להסב את המספר בן 32 הסיביות לכתובת לוגית זמנית. הדרך הטובה ביותר לנצל זיכרון פיסי זה היא עבור מאגר להחזקת נתונים, ואין להכיל בו אתתי RAM, או מבני נתונים קריטיים.

מרכיבי ה-device driver יכולים לבצע מיעון לזיכרון השמור ל-BIOS, בתחום הנמצא בין 640KB ל-1MB. מערכת ההפעלה אינה משתמשת במרחב כתובות זה כדי לשמור יישומים, או סגמנטים של המערכת. הדבר הראשון שה-device driver צריך לעשות הוא לקבל כתובת לוגית שתאפשר לו לפנות לנקודה מסוימת בזיכרון השמור. השלב הראשון בפעולה זו הוא חישוב של הכתובת הפיסית בת 32 הסיביות של היעד. לאחר מכן ה-device driver משתמש בשירות ה-DevHlp, PhysToVirt, כדי לקבל כתובת לוגית זמנית, שניתן להשתמש בה להזזת נתונים פנימה והחוצה מתוך שטח היעד. דוגמה לזיכרון כזה הוא המאגר (buffer) לתצוגת מונוכרום, אשר מתחיל ב-B000:0000h (או בכתובת פיסית 000B0000h). דוגמה אחרת לזיכרון כזה הוא זיכרון הנמצא במתאם, אבל ממופה לתוך האיזור השמור. ה-device driver יכול לתת ליישום, או לתת-מערכת ק/פ, אפשרות מיעון לזיכרון כזה בעזרת שירות ה-DevHlp, PhysToUVirt, היוצר ערך בורר לטבלת LDT. אולם, אין להשתמש בערך בורר זה בקריאות מערכת, מכיון שהוא אינו מצביע על זיכרון המערכת.

סינכרוניזציה

הסינכרוניזציה ב-device driver ב-OS/2 מתבטאת בשתי צורות: תיאום פעילויות בין מרכיבי ה-device driver עצמו, ותיאום פעילויות בינו לבין היישום. הסינכרון בין המרכיבים הפועלים בזמן משימה לבין מרכיבים הפועלים בזמן-פסק הוא יותר שאלה של "האם" או "מתי" ה-device driver צריך להחליט על עצירת thread של שיגרת האסטרטגיה, ומתי עליו להתחיל אותו שוב. התשובה לשאלות אלו תלויה בד"כ בצורך של ההתקן (ושל ה-device

driver) לנהל מספר חבילות בקשה שונות, ובדרך שבה היישום משתמש במישקים ל-device driver. באופן כללי, device driver יפעל לפי השיקולים הבאים:

- אם ההתקן תפוס, יש צורך להציב רק חבילות בקשה של READ ו-WRITE בתור הממתינים. בכל זמן שההתקן תפוס, שיגרת האסטרטגיה תציב חבילות של בקשות ק/פ ברשימה מקושרת ותחסום את ה-thread שהוציא את הבקשה. thread זה ימתין לשיגרת הטיפול בפסקי חומרה שתעבד את תור הממתינים. סוגים אחרים של חבילות בקשה מטופלים על-ידי שיגרת האסטרטגיה בצורה מיידית.
- בכל פעם שחבילת בקשה דורשת לקבל את מצב הק/פ להתקן, ה-device driver צריך לחסום את ה-thread בתוך שיגרת האסטרטגיה, ולהמתין לשיגרת הטיפול בפסקי חומרה שתעורר אותו לאחר שפעולת הק/פ הושלמה. בכל פעם שההתקן תפוס, הוא יכול להציב את חבילת הבקשה בתור הממתינים (כלומר ברשימה המקושרת של חבילות הבקשה). בקשות מעובדות לפי הסדר שנקבע על-ידי ה-device driver, כמו למשל ראשון-מגיע-ראשון-מקבל-שירות (FIFS), או לפי סדר אחר שנקבע על-ידי התהליך. ה-thread של שיגרת האסטרטגיה משתמש בשירות ה-DevHlp, Block, כדי להפסיק להתבצע. בכל פעם ששיגרת הטיפול בפסקי חומרה קובעת שפעולת ק/פ הסתיימה, היא יכולה להשתמש בשירות ה-DevHlp, Run, כדי לעורר את ה-thread שבשיגרת האסטרטגיה. לאחר מכן, שיגרת הטיפול בפסקי חומרה יכולה לקחת את הבקשה הבאה בתור ולשלוח אותה להתקן.
- תהליך יכול להעביר, על-ידי בקשת IOCtl, כתובת של מאגר הנמצא בסגמנט הנתונים של היישום. ה-device driver צריך לשמור שה-thread בתוך שיגרת האסטרטגיה ימשיך להתבצע, עד אשר הוא סיים להשתמש במאגר. שיגרת האסטרטגיה חייבת להשתמש בשירות ה-DevHlp, Lock, כדי להבטיח שהסגמנט של היישום לא יועבר ולא יוזז לפני שהיא קיבלה את הכתובת הפיסית. ה-device driver צריך גם לשמור את ה-thread של שיגרת האסטרטגיה בצורה כזו, שניתן יהיה לשחרר את הסגמנטים של היישום בעזרת שירות ה-DevHlp, Unlock. שיגרת הטיפול בפסקי חומרה תוכל לעורר את ה-thread של שיגרת האסטרטגיה כאשר הגישה לסגמנט הושלמה.
- אם שירותי ה-BIOS נמצאים בבעלות ה-device driver, צריך לארגן את הגישה להתקן בצורה סידרתית. כאשר ההתקן תפוס, שיגרת הטיפול בפסקי תוכנה מסמנת שק/פ של ה-BIOS ממתין בתור להתקן ושהיא ממתינה באתת, כדי לדעת מתי ניתן לבצע את הק/פ של ה-BIOS. הן שיגרת האסטרטגיה והן שיגרת הטיפול בפסקי חומרה יכולות לנקות את האתת כאשר הן רואות שק/פ ממתין ב-BIOS. בצורה זו הן מאפשרות לשיגרת הטיפול בפסקי תוכנה לדעת שה-BIOS יכול לגשת בצורה בטוחה להתקן.

אתתים הינם האמצעים העיקריים המשמשים לתקשורת בין ההתקן לבין היישום. אולם, ישנן הגבלות מסוימות שמכתיבות את הדרך שבה יכול ה-device driver להשתמש בהם. כפי שתזכור, יש שני סוגים של אתתים, אתת RAM ואתת מערכת. אתת RAM הוא מנגנון פשוט ולא מוגן המאפשר לתאם פעילויות. אתת מערכת הינו מנגנון המנוהל על-ידי מערכת ההפעלה ומוגן מפני מצבי נעילה ללא מוצא (לדוגמה, כאשר התהליך שבבעלותו נמצא האתת הפסיק לפעול מבלי לשחרר את האתת). הנקודות החשובות הקשורות לטיפול באתתים והמשפיעות על הפעילות ההדדית של ה-device driver עם היישום הן:

- device driver אינו יכול ליצור אתת מערכת, או להיות הבעלים שלו.
- device driver יכול לטפל באתת מערכת הנמצא בבעלות התהליך. התהליך חייב להעביר ל-device driver את המקשר לאתת המערכת כפקודת IOCtl. ה-device driver חייב להסב את המקשר שהוא קיבל למקשר שהוא יוכל להשתמש בו; הוא עושה זאת בעזרת שירות ה-DevHlp, SemHandle. לאחר מכן ה-device driver יכול להשתמש במקשר החדש שלו לאתת המערכת באמצעות שירותי ה-DevHlp, SemRequest ו-SemClear.
- device driver יכול להגדיר אתת RAM הנמצא בסגמנט הנתונים שלו. הוא גם יכול להכנס אליו בזמן-פסק.
- ה-device driver אינו יכול להשתמש בזמן-פסק באתת RAM הנמצא בסגמנט הנתונים של התהליך.
- device driver אינו מורשה לתת ליישום אפשרות מיעון לאתת RAM הנמצא בסגמנט הנתונים של ה-device driver, מכיון שהיישום לא יוכל להשתמש בערך בורר זה לטבלת ה-LDT.

הכוונה בכללים אלה היא שה-device driver חייב להשתמש באתת RAM, ולא באתת מערכת, כדי לתאם את הפעילויות בין המרכיבים השונים שלו. כמו כן, ה-device driver חייב להשתמש באתת מערכת הנמצא בבעלות היישום וסופק על-ידו, כדי להתקשר ישירות עם היישום.

קינון של פסקים

יש לשקול שימוש **בקינון** (nesting) פסקים לגבי שני המרכיבים של ה-device driver הפועלים בזמן-פסק, אם כי בדרך שונה לגבי כל אחד מהם. קינון של פסקים נוצר כאשר שיגרת הטיפול בפסקים מופעלת שוב בנקודת הכניסה שלה, ולפני שהיא הספיקה לחזור למי שקרא לה. נפשט ונאמר, ששיגרת הטיפול בפסקים קיבלה פסק חדש לפני שהיא סיימה את הטיפול בפסק הנוכחי. קינון של פסקים דורש עיבוד מיוחד והוא תלוי בגורמים אחדים, כמו:

- מתי שיגרת הטיפול בפסקי חומרה מודיעה על סוף פסק (EOI),
- מהי תדירות הפסקים בהתקן,
- אילו פסקי חומרה נוספים פעילים כרגע,
- מהי העדיפות היחסית של הפסק המבוקש ביחס לפסקים אחרים.

שיגרת הטיפול בפסקי חומרה היא החשופה ביותר לקינון פסקים, בהשוואה לשיגרת הטיפול בשעון העצר. שיגרת הטיפול בפסקי חומרה חייבת לקחת בחשבון את מגבלת הקינון של פסקים בגלל המחסנית המופעלת בזמן-פסק, אשר אינה יכולה לתמוך במספר אינסופי של פסקים. היכן שאפשר, שיגרת הטיפול בפסקי חומרה צריכה להגביל את מספר הפעמים שניתן להפעיל אותה ברצף, כדי לשמור על רמה נמוכה של שימוש במחסנית.

שיגרת הטיפול בשעון העצר חשופה גם היא לקינון של פסקים, אך בדרך לא ישירה בלבד. שיגרה זו פועלת בצורה יעילה לאחר שה-device driver של השעון שלח לשעון הודעת "סוף פסק", אשר מאפשרת פסק שעון נוסף בשעה שהשיגרה פועלת. שיגרת הטיפול בשעון רשאית להשלים את פעולתה ולחזור לתהליך שקרא לה לפני שניתן לקרוא לה שוב בנקודת הכניסה שלה. זאת אומרת, שפרק הזמן (הקבוע מראש) שמערכת ההפעלה צריכה להמתין לפני שהיא קוראת שוב לשיגרה יחלוף מספר פעמים, עד אשר היא תוכל לקבל אותה באמת. כדי לבדוק כמה זמן באמת עבר, שיגרת הטיפול בשעון יכולה להכנס למשתני

הזמן בסגמנט המידע הגלובלי (הכניסה לסגמנט מושגת בעזרת שירות ה-DevHlp, GetDOSVar).

ביצועי המערכת

ביצועי המערכת רגישים להתנהגות של מרכיבי ה-device driver. בגלל אופי הפעילויות שלו, ה-device driver חייב לפעול באותה רמה כמו מערכת ההפעלה, וביכולתו להשפיע על הדרך שבה מערכת ההפעלה מגיבה לאירועים. כדי לעשות אופטימיזציה של ההשפעות שיש ל-device driver על המערכת וכדי להעלות למקסימום את היכולת שלו להתקיים בסביבה רבת יישומים, הוא חייב לפעול בהתאם לכללים מסוימים:

- ה-device driver חייב למקם בסגמנט הנתונים שלו מבני נתונים, אשר קריטיים לביצוע שלו, ומאגרים המשמשים להעברת נתונים. מרכיבי ה-device driver יכולים לגשת מיידית לסגמנט הנתונים שלו, הן בזמן-פסק והן בזמן משימה.
- ה-device driver מעביר נתונים לסגמנט הנתונים של היישום, לסגמנט הנתונים שלו, או לגוש נתונים פיסי בזיכרון. הוא צריך לעשות זאת בבלוקים קטנים במטרה לשחרר את היע"מ מהר ככל האפשר. בזמן משימה, הוא חייב להשתמש בצורה מחזורית בשירותי ה-DevHlp, Yield ו-TCYield (בערך כל שלוש אלפיות השניה), כדי לאפשר ל-thread אחר להתבצע. בזמן-פסק עליו להגביל את זמן הביצוע שלו למינימום ולכן ברור, שאין זה מעשי להעביר כמויות גדולות של נתונים.
- מרכיבי ה-device driver חייבים להגביל את זמן העיבוד בזמן שפסקי החומרה מנותקים. ניתוק פסקי החומרה מאפשר לשלוט בגישה למבני נתונים קריטיים המשותפים לרכיבי זמן-פסק וזמן משימה של ה-device driver. אולם, הדבר משפיע על השירות שיקבלו פסקים של התקנים אחרים.
- ה-device driver חייב להקטין למינימום את זמן הביצוע שמוקדש לעיבוד פסק. שגרות הטיפול בפסקי החומרה ובשעון העצר חייבות לבצע מספר פעולות קטן ככל האפשר. הסיבה לכך היא שפעולות בזמן פסק מעכבות את התגובה של מערכת ההפעלה לאירועים אחרים בסביבה רבת המשימות. לדוגמה, אם שירות של פסק גורם לכך ש-thread קריטי מבחינת זמן הביצוע יוכן לריצה, על מערכת ההפעלה לאפשר את הביצוע שלו. אף על פי כן, הוא לא יכול להתבצע עד שמסתיים הביצוע של הפסק המקבל כעת שירות.

בנוסף לכך, שיגרת הטיפול בפסקי חומרה צריכה לשלוח הודעת "סוף פסק" להתקן שלה, קרוב ככל האפשר לסיום העיבוד הקריטי של הפסק. הדבר מאפשר לכל שאר הפסקים הנמצאים בבקר הפסקים להשלח ליע"מ לשם קבלת שירות.

תיחול של DEVICE DRIVERS

מערכת ההפעלה OS/2 מכירה ב-device driver ובהתקנים שהוא תומך בהם רק לאחר הטעינה והתיחול שלו שנעשים בזמן העיבוד של משפטי =DEVICE בקובץ ה-CONFIG.SYS. כל משפט =DEVICE מעובד לפי הסדר שבו הוא מופיע ב-CONFIG.SYS. מערכת ההפעלה טוענת את קובץ התכנית של ה-device driver המזוהה על-ידי המשפט =DEVICE, וקוראת לשיגרת האסטרטגיה עם חבילת הבקשה של הפקודה INIT. זו מודיעה ל-device driver לתחל את עצמו ואת ההתקן שלו.

תיחול ה-device driver נעשה בתהליך מיוחד במערכת ההפעלה ולכן שלב ביצוע זה נקרא "מצב תיחול" (init mode). בתהליך מיוחד זה של מערכת ההפעלה, ה-device driver מתבצע כ-thread של תהליך התיחול במצב מוגן עם רמת הרשאה של יישום. זאת אומרת, שה-device driver רשאי להפעיל פונקציות קישור דינמי מסוימות, כמו מישקים של מערכת הקבצים, או אמצעים לטיפול בהודעות. "מצב תיחול" הינו הפעם היחידה שהוא רשאי להפעיל פונקציות מסוג זה. ה-device driver יכול לבצע ק/פ לקבצים, לשם קריאה מתוך קובץ המכיל גופנים (פונטים) של המסך, או מידע על התצורה של ההתקן. הוא יכול גם להשתמש באמצעים לטיפול בהודעות (DosGetMessage, DosInsMessage ו-DosPutMessage) כדי להציג הודעות למשתמש על מצב ה-device driver וההתקן. למרות שהתיחול של ה-device driver מתבצע ברמת הרשאה של היישום (3), יש לו רמת הרשאת ק/פ (2) ועל-כן הוא יכול לגשת לכניסות הק/פ של ההתקן.

ה-device driver נמצא בזיכרון שמתחת ל-640BK ולכן הוא תופס חלק ממרחב הכתובות השמור לסביבת DOS. כדי לצמצם את צריכת הזיכרון בתחום זה, ה-device driver יכול להפריד את הפקודות והנתונים המשמשים לתיחול מאלה המשמשים לפעולות רגילות. בדרך זו ניתן לבטל בסיום התיחול של ה-device driver את הסגמנטים המשמשים לתיחול בלבד. ה-device driver יכול גם להטעין סגמנטים בגודל מקסימלי (64KB) ולהחזיר שטחים אלה למערכת כאשר אין לו צורך בהם. ה-device driver הינו מודול קטן הנמצא מתחת ל-640KB, אבל יש לזכור שביכולתו להקצות ולהשתמש בזיכרון נוסף באמצעות שירותי ה-DevHlp לניהול הזיכרון.

הדוגמה הבאה מתארת את שלבי התיחול.

1. מערכת ההפעלה קוראת לשיגרת האסטרטגיה בעזרת מחוון (ES:BX) לחבילת הבקשה של הפקודה INIT. שיגרת האסטרטגיה מתבצעת כ-thread תחת תהליך התיחול ובתוך מחסנית של התהליך, ויש לה אפשרות מיעון לסגמנט הנתונים של ה-device driver (באוגר DS). היא מתבצעת ב-CPL=3, אבל עם הרשאת ק/פ (IOPL).
2. שיגרת האסטרטגיה שומרת את המחוון למישק ה-DevHlp שסופק על-ידי חבילת הבקרה INIT למטרות של שימוש עתידי.
3. שיגרת האסטרטגיה משיגה את המשתנים שצוינו במשפט =DEVICE והועברו בחבילת הבקשה INIT.
4. שיגרת האסטרטגיה קובעת את התצורה של ההתקן ומזהה את רמת הפסק, את כניסות הק/פ ואת מרחב הכתובות שישמש את ההתקן.
5. שיגרת האסטרטגיה רושמת נקודות כניסה נוספות לתוך ה-device driver. למשל, עבור שיגרת הטיפול בפסקי חומרה, שיגרת הטיפול בשעון העצר ושיגרת הטיפול בפסקי תוכנה.

6. שיגרת האסטרטגיה מתחלת את ההתקן. אם התיזול נכשל, ה-device driver חייב לבטל את הרישום של נקודות הכניסה ולשחרר זיכרון שהוקצה.
7. לאחר שסיימה את התיחול, שיגרת האסטרטגיה מציבה את קוד הסיום של הפעולה בחבילת הבקשה, מצביעה על הכתובות היחסיות המסיימות את הסגמנטים של הפקודות והנתונים ולבסוף חוזרת למערכת ההפעלה.

BIOS מתקדם ושיתוף פסקים

המערכת האישית PS/2, דגמים 50, 60, 70 ו-80 מספקת שני מאפיינים חשובים לשם ניהול ההתקן ב-OS/2. מאפיינים אלה אינם קיימים ב-PC-AT וב-PC-XT מודל 286. המאפיינים הם מישק BIOS מתקדם (Advanced BIOS) וסביבה של פסקי חומרה הרגישים לרמת סיווג. ה-BIOS המתקדם חשוב, מאחר והוא מספק ל-device driver ב-OS/2 מישק ביניים להתקן, אשר משחרר את ה-device driver מתלות במאפיינים מסוימים של התקן. כאשר device driver אינו נכתב עם מישק ביניים, הוא מכיל בתוכו את פרטי התפעול המפורטים של ההתקן, ועל כן הוא מוגבל במידת החופש לקבל שינויים בהתקן זה. הסביבה של פסקי חומרה הרגישים לרמת סיווג מתאימה יותר לשיתוף פסקי חומרה מאשר הסביבה של פסקי חומרה מופעלי סף (edge triggered hardware interrupts), הקיימת במחשבים אישיים (PC).

BIOS מתקדם - ABIOS

BIOS מתקדם (Advanced BIOS - ABIOS) הינו רובד של מישק להתקן שיש לו מספר מאפיינים המבדילים אותו מה-BIOS:

- ABIOS מופעל על-ידי קריאה רחוקה ולא על-ידי פסק חומרה.
- ABIOS המתקדם יכול להתבצע הן במצב אמיתי והן במצב מוגן.
- ABIOS המתקדם מקבל את הפעולות שהוא צריך לבצע ואת הפרמטרים שהוא צריך להשתמש בהם ממבנה נתונים הנקרא גוש הבקשה (request block). ה-device driver קובע את גוש הבקשה ומעביר מחוון למבנה הנתונים בכל פעם שהוא קורא ל-ABIOS.
- ABIOS מזהה התקן מסוים עם מקשר הנקרא מספר מזהה לוגי (Logical ID - LID) ועם מספר יחידה הנמצא באותו LID.
- ק/פ להתקן מבוצע על-ידי קריאה לאחת מנקודות הכניסה של ה-ABIOS: נקודת הכניסה START מתחילה ביצוע של פונקציה, נקודת הכניסה INTERRUPT מטפלת בפעולה בזמן-פסק ונקודת הכניסה TIMEOUT משמשת לעיבוד פסק-זמן.
- פונקציית ק/פ יכולה להיות מסונכרנת, או מבוצעת בשלבים. בקשת ק/פ מסונכרנת חייבת לרוץ עד לסיומה לפני שהיא חוזרת לתהליך שביקש אותה. בקשת ק/פ בשלבים מחייבת סידרה של פעולות. בכל אחד מהשלבים, נדרש ה-thread הקורא לבצע פעולה שתעביר את ביצוע הבקשה לשלב הבא.
- העברת נתונים יכולה להיות מבוססת על כתובת לוגית או על כתובת פיסית.
- device driver נכנס למישק ה-ABIOS דרך שירותי ה-DevHlp: GetLIDEntry, FreeLIDEntry, ABIOSCall ו-ABIOSCommonEntry.

כדי שה-device driver יוכל להשתמש בהתקן באמצעות ABIOS, עליו לזהות את ההתקן באמצעות מקשר להתקן - מספר מזהה לוגי (LID). הוא עושה זאת בעזרת שירות ה-DevHlp, GetLIDEntry. ABIOS מזהה התקנים בעזרת LIDs ו-OS/2 עושה זאת בעזרת כותרות ההתקנים. לכן ה-device driver צריך לבצע התאמה בין הדרך שההתקן מזוהה על-ידי מערכת ההפעלה לבין הדרך שבה הוא מזוהה על-ידי מישק ABIOS. יש למלא אחר מספר כללים בזמן שנעשית התאמה בין ההתקן לבין מקשר ABIOS להתקן, מכיון שמקשר (LID להלן) יכול לזהות התקן אחד או קבוצה של התקנים. הכללים הם:

- device driver להתקני תו עם כותרת התקן אחת: התקן התו ממופה ליחידה הראשונה של ה-LID. אין משתמשים ביחידות אחרות ב-LID זה והן אינן עומדות לרשות device drivers אחרים.
- device driver להתקני תו עם יותר מכותרת התקן אחת: התקן התו הראשון ממופה ליחידה הראשונה ב-LID, התקן התו השני ממופה ליחידה השניה ב-LID, וכן הלאה. אם אין ב-LID מספיק יחידות עבור כל ההתקנים הנתמכים על-ידי ה-device driver, על ה-device driver להשיג LID נוסף ולהמשיך במיפוי של ההתקנים ליחידות ב-LID.
- device driver של התקן בלוקים עם יחידה אחת, או יותר, שצוינה בכותרת ההתקן שלו, חייב למפות את היחידה הראשונה של כותרת ההתקן ליחידה הראשונה של ה-LID וכן הלאה עד אשר הוא ממפה את כל היחידות שהוא תומך בהן. מיפוי זה עלול לדרוש מה-device driver להשיג יותר מ-LID אחד.

כל LID שלא נוצל על-ידי device driver אחד יכול לעמוד לרשות אחר. כאשר device driver יכול לשחרר את ההתקן, הוא יכול לשחרר את מקשר ה-ABIOS להתקן בעזרת שירות ה-DevHlp, FreeLIDEntry.

לאחר שה-device driver השיג את ה-LID, הוא קובע אילו פרמטרים הוא צריך להעביר למישק ABIOS לגבי ההתקן המסוים הזה. כדי להשיג מידע זה, עליו לקבוע מבנה נתונים באורך קבוע הנקרא גוש הבקשה. גוש זה מכיל LID וקוד ביצוע, והוא מחזיר את הפרמטרים של ה-LID. ה-device driver יכול להשתמש בשירותי ה-DevHlp, ABIOSCall או ABIOSCommonEntry, כדי לקרוא לנקודת הכניסה START המתייחסת ל-LID זה. ABIOS מחזיר מידע אשר כולל:
- רמת הפסק שההתקן מכוון אליה.
- רמת הבוררות (arbitration level), אם היא ישימה.
- מספר היחידות המכוסות על-ידי ה-LID.
- האורך של גוש הבקשה של פונקציות אחרות המסופקות על-ידי ה-ABIOS.
- המאפיינים של מחווני הנתונים (כמו למשל, לוגי מול פיסי).

ה-device driver לומד ממידע זה:
- איזה פסק חומרה הוא צריך לרשום שיגרה לטיפול בפסקי חומרה.
- איזה סוגים של מחווני נתונים מצפה מישק ABIOS לקבל.
- מהו גודל השטח שצריך להשמר בסגמנט הנתונים של ה-device driver לצורך גושי הבקשות של ABIOS.

כדי להפעיל פונקציית ABIOS מסוימת, ה-device driver חייב להכין תחילה גוש בקשה המכיל LID וקוד ביצוע. לאחר מכן הוא משתמש באחד משירותי ה-DevHlp, ABIOSCall או ABIOSCommonEntry, כדי לקרוא לאחת משלוש נקודות הכניסה ל-ABIOS המתאימים ל-LID זה. ABIOS מבצע את הפעולה המבוקשת ובסיומה מציב את המצב בגוש הבקשה. כאשר שירותי ABIOSCall או

ABIOSCommonEntry חוזרים ל-device driver, הוא חייב לבדוק את גוש הבקשה כדי לקבוע מה היתה התוצאה.

השתמשנו בדוגמה הבאה כדי לתאר איך ה-device driver משתמש ב-ABIOS. הדוגמה מניחה שהוא ביצע את הפעולות הבאות: השיג את ה-LID, שמר שטח לגושי הבקשה של ABIOS, זיהה את פסק החומרה ורשם את שיגרת הטפול בפסקי החומרה. להלן שלבי הביצוע:

1. מערכת ההפעלה קוראת לשיגרת האסטרטגיה עם חבילת בקשה המכילה בקשת ק/פ להתקן היעד.
2. אם ההתקן פנוי, שיגרת האסטרטגיה קובעת את גוש הבקשה עם ה-LID, את מספר הפונקציה וכל פרמטר אחר הנדרש על-ידי ה-ABIOS, ותיחול של שדה הקוד החוזר ל-FFFFh. שיגרת האסטרטגיה משתמשת באחד משירותי ה-DevHlp, ABIOSCall או ABIOSCommonEntry, כדי להפעיל את נקודת הכניסה START שתתחיל את ביצוע הפונקציה. אם ההתקן תפוס, שיגרת האסטרטגיה יכולה להציב את חבילת הבקשה בתור הממתינים. שיגרת הטיפול בפסקי חומרה תוכל להפעיל את נקודת הכניסה כדי שתתחיל את ביצוע הפונקציה.
3. לאחר החזרה משירות ABIOSCall או ABIOSCommonEntry, שיגרת האסטרטגיה בודקת את שדה הקוד החוזר בגוש הבקשה, כדי לקבוע מה היתה התוצאה של פונקציית ה-ABIOS.
   - אם הקוד החוזר מצביע על שגיאה, שיגרת האסטרטגיה מציבה בשדה המצב שבחבילת הבקשה קוד המסמן שגיאה בהתקן, וחוזרת למערכת ההפעלה.
   - אם הקוד החוזר מצביע על כך שהפונקציה הסתיימה בהצלחה, שיגרת האסטרטגיה מציבה הודעה על כך בשדה המצב של חבילת הבקשה. לאחר מכן היא חוזרת למערכת ההפעלה.
   - אם הקוד החוזר מצביע על כך שהפונקציה מוכנה לביצוע השלב הבא (כלומר, היא ממתינה שיתרחש פסק בהתקן), שיגרת האסטרטגיה חוסמת את ביצוע ה-thread בזמן שהיא ממתינה לפסק בהתקן.

   שיגרת האסטרטגיה חייבת לבדוק אות-סימון (flag) המשמש גם את שיגרת הטיפול בפסקי חומרה, לפני שהיא חוסמת את ה-thread. הסיבה לכך היא, שהפסק בהתקן יכול להתרחש לאחר שה-ABIOS מעדכן את שדה הקוד החוזר בגוש הבקשה אבל לפני ששיגרת האסטרטגיה הספיקה לבדוק את הקוד החוזר. במקרה זה, שיגרת הטיפול בפסקי חומרה נקראת כתוצאה מפסק בהתקן, נותנת שירות לגוש הבקשה ומשלימה את הבקשה. לאחר מכן, אות הסימון שנקבע על-ידי שיגרת הטיפול בפסקי חומרה מודיע לשיגרת האסטרטגיה שהבקשה הושלמה כבר, ולכן עליה לשלוח את חבילת הבקשה למערכת ההפעלה ולא לחסום את ה-thread של היישום.

   לגבי בקשה בשלבים, נקודת הכניסה START מציבה קוד חוזר בגוש הבקשה, כדי שיצביע על כך שהפעולה לא הושלמה. מספר בקשות לא גמורות (outstanding requests) השייכות לאותו LID, יכולות להמתין בו-זמנית. ברגע שבקשה הותחלה היא מוכנה לשלב הבא. בשלב זה היא עדיין נחשבת לבקשה לא גמורה אפילו אם שירות START לא חזר ל-device driver שקרא לו, כפי שהוצג בתרחיש שתואר קודם.
4. שיגרת הטיפול בפסקי חומרה נקראת כתוצאה מפסק בהתקן.

5. שיגרת הטיפול בפסקי חומרה חייבת לעבד את כל גושי הבקשה הלא-גמורים בכל LID הקשור לפסק זה (יכול להיות יותר מאחד). היא חייבת לקרוא לנקודת כניסה של ABIOS כדי לקבל שירות INTERRUPT לכל גוש בקשה שהמתין לפסק בהתקן, במטרה לתת ל-LID שירות מלא. השיגרה חייבת להכין את עצמה למצב שבו שירות START עידכן את הקוד החוזר בגוש הבקשה כדי להראות שהוא ממתין לפסק, אך לא חזר ל-device driver שקרא לו. שיגרת הטיפול בפסקי חומרה חייבת לחפש גושי בקשה הנמצאים במצב זה ולכלול אותם ברשימה של אלה שצריכים לקבל שירות בזמן-פסק.
6. אם הפסק בהתקן נגרם על-ידי אחד מגושי הבקשות השייכים ל-LID מסוים, שיגרת הטיפול בפסקי חומרה קוראת לנקודת הכניסה INTERRUPT, כדי לשרת את שאר גושי הבקשות השייכים לאותו LID.
7. אם ל-LID אין אף לא גוש בקשה אחד שממתין לפסק, שיגרת הטיפול בפסקי חומרה קוראת לנקודת הכניסה INTERRUPT, כדי שתבצע פונקציה המספקת שירות ברירת מחדל לפסקים.

שיגרת הטיפול בפסקי חומרה מודיעה לבקר הפסקים על סוף פסק, רק לאחר שהיא שירתה את כל גושי הבקשות הלא-גמורים השייכים לאותו LID שגרם את הפסק בהתקן. אם השיגרה רואה שהפונקציה המבוקשת הושלמה, היא יכולה להתחיל את גוש הבקשה הבא המבוסס על חבילת הבקשה הבאה הממתינה בתור.

לאחר ששיגרת הטיפול בפסקי חומרה השלימה את פעולתה, היא חוזרת למערכת ההפעלה.

שיתוף בפסקי חומרה

שיתוף בפסק חומרה בין מספר התקנים אפשרי רק כאשר ההתקנים תוכננו מראש לקבל שיתוף פסקים. התקן חייב לדעת אם הוא הפיק פסק, כדי שהוא יוכל לשתף התקנים אחרים בפסק שלו. הדבר מאפשר לשיגרת הטיפול בפסקי תוכנה לבדוק את ההתקן כדי לקבוע אם הוא דורש שירות. ההתקן יכול לשחרר את חסימת הפסקים, כאשר הפסק שלו מקבל שירות משיגרת הטיפול בפסקי תוכנה.

בדגמים 50, 60, 70 ו-80 של PS/2, פסקי החומרה רגישים לרמת סיווג והם אינם מופעלי-סף. שני המונחים, "רגישים לרמת סיווג" ו"מופעלי-סף", מתייחסים לשיטה שבה בקר הפסקים משתמש כדי לקבוע אם ההתקן הפיק פסק. כאשר בקר הפסקים זיהה פסק מהתקן, הוא שולח את הפסק ליע"מ, אשר משתמש בטבלת הווקטורים של הפסק כדי להפעיל את שיגרת הטיפול בפסקי תוכנה.

בקר הפסקים מזהה פסקים מופעלי-סף בהתקן על-ידי זיהוי של שינוי באות, כמו מעבר של אות ממצב נמוך לגבוה. האות יכול להשאר ברמת הגבוהה, ואז ניתן לאתחל אותו מבלי לגרום לבקר הפסקים לחשוב שהופק פסק נוסף בהתקן. השיטה של מעבר סף מסוים, היא שמודיעה לבקר הפסקים על פסק בהתקן.

בקר הפסקים מזהה פסקים הרגישים לרמת סיווג בהתקן על-ידי זיהוי של רמה מסוימת, או ערך מסוים באות. כל זמן שהאות נשאר באותה רמה, בקר הפסקים מאמין שיש צורך לשרת את הפסק. רק כאשר יתבצע אתחול (reset) של האות מרמה זו, יפסיק בקר הפסקים לנסות ולהתמודד עם הפסק. במלים אחרות, לגבי פסקים הרגישים לרמת סיווג, יש לבצע את האתחול של הפסק בהתקן. בדרך זו ההתקן יאתחל את האות שלו לבקר הפסקים. שיגרת הטיפול בפסקי תוכנה חייבת להודיע על סוף פסק לאחר שהוסר מצב הפסק בהתקן. יכול לקרות מצב שבו

הודעת הסוף פסק נשלחה לבקר הפסקים לפני שהתבצע אתחול של מצב הפסק. במקרה כזה האות יהיה עדיין ברמה שמגדירה פסק לבקר הפסקים וכתוצאה מכך בקר הפסקים ישלח שוב את הפסק ליע"מ. בתרשים 50 תמצא הסבר של השלבים בתהליך זה.

בדגמים 50, 60, 70 ו-80, לכל פסקי החומרה יש אפשרות להיות משותפים למספר התקנים. עם זאת, ישנם מספר מצבים שבהם שיתוף פסקים אינו מעשי. לדוגמה, OS/2 אינה מאפשרת למספר שגרות המטפלות בפסקים להתחלק בפסק של שעון המערכת (IRQ O) מכיון שהוא חייב לעמוד לרשות יישום DOS בסביבת DOS. דבר דומה קורה כאשר ה-device driver ב-OS/2 משתמש ב-BIOS בסביבת DOS כדי לבצע עיבוד בזמן-פסק שיתמוך בק/פ של יישום DOS. במקרה זה אסור ל-device driver לשתף אחרים בפסק, כי ה-BIOS מניח שהוא הבעלים של עיבוד הפסק. בנוסף לכך ההתקן עצמו יכול למנוע שיתוף בפסק שלו. למשל, אם ההתקן מפיק פסקים לפני ששיגרת הטיפול בפסקי חומרה הותקנה, או שלא ניתן להפסיק בו את הפקת הפסקים, לא ניתן להתחלק בפסק עם התקן אחר. אם התקן מפיק פסק לפני שהשיגרה לטיפול בפסקים הותקנה, אז OS/2 חייבת למסך את הפסק הזה, כי אין שום דרך לאתחל את מצב הפסק בהתקן. אם ההתקן אינו מפסיק להפיק פסקים, אפילו אם השיגרה לטיפול בפסקים חדלה לפעול, OS/2 חייבת למסך את הפסק כי אין אף מנגנון שיכול לאתחל את ההתקן.

אם ההתקן וה-device driver לא יהיו מתוכננים לקבל שיתוף פסקים, לא ניתן יהיה לממש שיתוף פסקים בין מספר התקנים. כאשר ה-device driver רושם את נקודת הכניסה לשיגרת הטיפול בפסקי חומרה, הוא חייב להצהיר הן על סוג הפסק והן על היכולת להתחלק בפסק. אם ה-device driver מציין שהוא אינו רוצה להתחלק בפסק, OS/2 נותנת לו זכויות בלעדיות עליו. אם ה-device driver מציין שהוא יתחלק בפסק, OS/2 מאפשרת רישום של שגרות אחרות לטיפול בפסקי חומרה המצביעות על כך שהן רוצות להתחלק בפסק. אפשר להגיד ששיתוף פסקים דורש רמה מסוימת של שיתוף פעולה מצד ה-device driver. ניתן גם להבין שהן ה-DOS device driver והן יישום ה-DOS אינם יכולים להתחלק בפסקי חומרה, במיוחד מאחר שביצוע בסביבת DOS מושעה בכל פעם שהמפעיל מעביר את סביבת ה-DOS לרקע.

1. התקן אחד או יותר מאתחלים את אות הפסק. בצורה זו הם מתכוננים להודיע לבקר הפסקים אודות פסק.
2. התקן אחד או יותר מבצעים אות פסק ברמה שמזהה את הפסק לבקר הפסקים.
3. בקר הפסקים שולח את הפסק ליע"מ.
4. היע"מ גורמת לביצוע של שיגרת הטיפול בפסקים.
5. שיגרת הטיפול בפסקים בודקת את ההתקן כדי לקבוע אם יש בו פסק שממתין לטיפול. אם נכון הדבר, היא משרתת את ההתקן אשר מאתחל את מצב הפסק בתוכו (ועל-ידי כך גורם לאתחול האות בבקר הפסקים). לאחר מכן היא שולחת הודעת סוף פסק לבקר הפסקים. לאחר קבלת "סוף פסק", בקר הפסקים שולח את הפסק ליע"מ אם אות הפסק עדיין מצביע על כך שפסק ממתין. אם להתקן אחר יש פסק שלא קיבל שירות כאשר נשלחה הודעת "סוף פסק", ההתקן ממשיך לשלוח לבקר הפסקים את האות ברמה שמזהה פסק.

תרשים 50. פסקים רגישים לרמת סיווג

שגרות שונות לטיפול בפסקי חומרה, הרוצות להתחלק באותו פסק, חייבות למלא אחר נהלים מסוימים. שיגרת הטיפול בפסקי חומרה אחראית לתיחקור ההתקן שלה כדי לבדוק אם הוא גרם לפסק. אם ההתקן לא גרם לפסק, שיגרת הטיפול בפסקי חומרה חייבת לחזור למערכת ההפעלה ולהצביע על כך שהפסק אינו שייך להתקן שלה. בנוסף לכך, אסור לשיגרת הטיפול בפסקי חומרה לשלוח הודעת "סוף פסק" כי היא לא נתנה שירות להתקן שלה. אם ההתקן שלה גרם את הפסק, השיגרה חייבת לבצע את הפעולות הבאות:

- לשרת את ההתקן שלה.
- לאתחל את מצב הפסק בהתקן.
- לשלוח הודעת "סוף פסק" מהר ככל שניתן.
- להודיע למערכת ההפעלה, כאשר היא חוזרת אליה, שהיא הבעלים של הפסק.

מערכת ההפעלה מנהלת רשימה של שגרות לטיפול בפסקים עבור כל פסק חומרה. מערכת ההפעלה תקרא לכל שיגרה ברשימה, עד אשר היא תמצא את זו שתטען לבעלות על הפסק. אם לא נמצאה שיגרה כזו, OS/2 תמסך את הפסק. הסיבה לכך היא שאין כל מנגנון שהוא שיכול לאתחל מצב הפסק של פסקים רגישים לרמת סיווג, וללא שימוש באמצעי קיצוני זה, בקר הפסקים היה ממשיך לשלוח את הפסק למערכת.

איך ליצור DEVICE DRIVER המתאים ל-OS/2

תכנית device driver שונה מתכנית יישום במספר נקודות, וכוללת את המאפיינים הבאים:

- אינה יכולה להיות תהליך בפני עצמו.
- נמצאת ברמת הרשאה של גרעין מערכת ההפעלה, ולא ברמת ההרשאה של היישום.
- אינה יכולה לבצע קריאות לפונקציות קישור דינמיות.
- יכולים להיות בה סגמנט נתונים אחד וסגמנט פקודות אחד.
- הסגמנטים חייבים להיות מקושרים, ולהופיע בסדר מסוים בקובץ ה-EXE.
- אינה מספקת את סגמנט המחסנית שלה.
- הסגמנטים חייבים להיות מקושרים כספריה ולא כיישום.
- יכולה לקבל סגמנטים נוספים בתנאים מסוימים.
- חייבת להכיל כותרת התקן אחת לפחות.
- כותרת ההתקן חייבת להיות האובייקט הראשון המוגדר בסגמנט הנתונים.

device driver אינו יכול להיות תהליך בפני עצמו, והוא פועל בשמו של התהליך ביישום. בזמן משימה, כאשר הוא מטפל בחבילת הבקשה שהופקה על-ידי בקשת הק/פ של היישום, ה-device driver מתבצע כתת-שיגרה שלו. כאשר device driver מטפל בפסק של התקן, הוא מתבצע מחוץ לגבולות של התהליכים במערכת. פרט למאפיינים אלו יש לו מספר מאפיינים של תהליך:

- יש לו אפשרות מיעון לסגמנטים של הפקודות והנתונים שלו. יש לציין שהדבר נעשה דרך טבלאות GDT, ולא דרך טבלאות LDT.
- יכול להיות הבעלים של זיכרון, אך זה זיכרון פיסי ולא זיכרון המחולק לסגמנטים.

- יכול להיות בעלים של אתתי RAM, כי אלה יכולים להיות מוגדרים בסגמנט הנתונים שלו.

פרט למאפיינים אלה הדומים לאלה של תהליך, אפשר להגיד שה-device driver דומה יותר למודול ספריה ולא לתכנית יישום.

תכנית device driver פועלת ברמת הרשאה של גרעין מערכת ההפעלה ולא של היישום, והיא מותקנת כחלק של מערכת ההפעלה בזמן התיחול הכללי של המערכת. לאחר מכן מערכת ההפעלה קוראת לתכנית device driver כדי שתבצע משימות ק/פ בשמו של היישום. למעשה ניתן לראות בתכנית זו הרחבה של מערכת ההפעלה.

device driver אינו יכול לבצע קריאות לפונקציות קישור דינמיות. הגבלה זו מוטלת עליו כי הוא אינו תהליך ואינו נמצא ברמת הרשאה של היישום. אולם ישנו פרק זמן שבו ה-device driver יכול לבצע קריאות לפונקציות קישור דינמיות בקבוצה מוגבלת של מישקי קישור דינמיים. דבר זה מתרחש בזמן התיחול של ה-device driver. בזמן שמערכת ההפעלה קוראת לו עם חבילת הבקשה INIT, הוא מתבצע למעשה כתהליך מיוחד של המערכת. הוא מקבל מעמד זמני מיוחד הדומה ליישום, וכך הוא יכול להשתמש במישקי התכנות המאפשרים קישור דינמי.

device driver יכול להכיל רק סגמנט פקודות אחד וסגמנט נתונים אחד, אשר הגודל של כל אחד מהם יכול להגיע עד ל-64KB. הטכניקה של שמירת כל הפקודות בסגמנט אחד (ושימוש בסגמנט יחיד לנתונים) ידועה בשם "המודל הקטן". במלים אחרות, ה-device driver מבצע קריאות קרובות משיגרה אחת לאחרת.

הסגמנטים של device driver חייבים להיות מקושרים, כך שהם יופיעו בסדר מסוים בקובץ EXE. המערך של קובץ EXE מתחיל בכותרת שבעקבותיה באים הסגמנטים. סגמנט הנתונים חייב להופיע ראשון ובעקבותיו בא סגמנט הפקודות. סידור זה דומה לזה הנהוג בתכנית device driver ב-DOS כאשר ההבדל היחיד הוא בכך שהתכנית ב-OS/2 היא קובץ EXE. בתרשים 51 תמצא שרטוט של מערך קובץ ה-EXE של device driver.

כותרת הקובץ EXE
סגמנט נתונים
סגמנט פקודות
סגמנט נוסף
:
סגמנט נוסף

תרשים 51. מערך קובץ ה-EXE של ה-device driver

לתכנית device driver אין סגמנט מחסנית, כי היא פועלת בדומה לתת-שיגרה: משתמשת במחסנית של התכנית שקראה לו. מערכת ההפעלה מספקת מחסניות לפעולות הן בזמן-משימה והן בזמן-פסק. שיגרת הטיפול בפסקי תוכנה רצה במחסנית של יישום ה-DOS. בכל המקרים, device driver חייב לדאוג שהשימוש במחסנית יהיה מינימלי.

הסגמנטים של device driver חייבים להיות מקושרים כספריה ולא כיישום. תכנית device driver דומה לתת-שיגרה, אשר פועלת מטעמו של ה-thread שקרא לה וגם משתמשת במחסנית שלו. בהקשר זה, לתכנית device driver ב-OS/2 יש מאפיינים הדומים לחבילת ספריה. תכנית הקישור מכירה שני סוגים של תכניות, יישומים וספריות. ה-device driver דומה יותר לתכנית ספריה מאשר לתכנית יישום.

תכנית device driver יכולה להכיל סגמנטים נוספים מלבד אלה של הפקודות והנתונים, אבל רק בתנאים מסוימים. במקרים אלה, יש צורך למקם את הסגמנטים הנוספים בסוף קובץ ה-EXE, לאחר סגמנט הפקודות (ראה תרשים 51). ה-device driver יכול להתייחס לסגמנטים אלה רק בזמן התיחול שלו, כלומר רק בזמן שהוא מעבד את חבילת הבקשה INIT. לאחר שהוא חוזר מהתיחול, מערכת ההפעלה מבטלת את הסגמנטים הנוספים ושומרת רק את שני הראשונים, סגמנט הפקודות וסגמנט הנתונים. ה-device driver משתמש בסגמנטים הנוספים כסגמנטים לנתונים, אשר מכילים בד"כ הודעות למפעיל. בזמן התיחול, ה-device driver יכול להשתמש באמצעים לטיפול בהודעות כדי להציג הודעות למפעיל כדי לציין למשל אם התיחול עבר בהצלחה, או לא.

בתכנית device driver חייבת להיות כותרת התקן אחת לפחות, אשר מזהה את ההתקן ל-OS/2. המבנה שלה דומה לכותרת ההתקן הנמצאת ב-device driver ב-DOS. יש מספר שדות בכותרת ההתקן שיש צורך להגדיר. בתרשים 52 תראה תרשים של כותרת ההתקן ב-OS/2.

יש צורך לתחל את **הכותרת הבאה** (Next Header) בערך דמה (null) (1- או FFFFh).

0 1 2 3	4 5	6 7	8 9	A B C
הכותרת הבאה	מאפיין ההתקן	כתובת חסית של שיגרת האסטרטגיה	שמור	שם...

D E F 10 11	12 13 14 15 16 17 18 19
..... או יחידות	שמור

תרשים 52. כותרת ההתקן

מאפיין ההתקן (Device Attribute) מציין את התכונות של ההתקן וכולל בין השאר:

- אם ההתקן הוא התקן בלוקים או התקן תו.
- אם ההתקן הוא התקן בלוקים,
  - האם ה-device driver משתמש בפרמטר הבלוק של ה-BIOS, או במתאר אחר כדי לזהות את המצע המגנטי.
  - האם ה-device driver תומך במצע נתיק.
- אם ההתקן הוא התקן תו,
  - האם שם ההתקן מוגן על-ידי מערכת הקבצים.
  - האם ה-device driver רוצה שמערכת הקבצים תעביר לו בקשות OPEN ו-CLOSE.
- האם ה-device driver שהוא הבעלים של ההתקן שייך ל-OS/2.
- האם ההתקן הוא התקן מערכת, כמו שעון, או שהוא פלט או קלט סטנדרטי.

הכתובת היחסית של שיגרת האסטרטגיה (Strategy Offset) הינה מתחילת סגמנט הפקודות עד לנקודת הכניסה לשיגרת האסטרטגיה. **שם או יחידות** (Name or Units), מכיל את השם של התקן התו או התקן הבלוקים הנתמך על-ידי ה-device driver. השם חייב להיות מיושר לשמאל (כלומר מתחיל בכתובת היחסית Ah). שם של התקן תו חייב להיות כמו שם קובץ, במחרוזת ASCII ומוגבל לשמונה תווים (לא כולל ההרחבה). אולם, לשם של ההתקן יש עדיפות על השם של הקובץ בכל הנוגע לפתיחת אובייקט בעל שם (עם DosOpen). זאת אומרת, שלא ניתן לקובץ שם זהה לשם של התקן התו, כי מערכת הקבצים תפתח תמיד את ההתקן לפני שהיא תנסה לפתוח את הקובץ. כדי למנוע התנגשות בין שמות הקבצים לבין שמות התקני התו, יש לתת להתקן שם המכיל תווי ASCII שאינם שכיחים, כמו הסימן "$", למשל.

כותרת ההתקן חייבת להיות מבנה הנתונים הראשון בסגמנט הנתונים. במלים אחרות, היא חייבת להתחיל בכתובת יחסית 0 בסגמנט הנתונים של ה-device driver. בזמן שתכנית device driver מוטענת, OS/2 בוחנת את כותרת ההתקן, הערוכה כמישק עבור למערכת ההפעלה.

device driver תומך בד"כ בהתקן אחד, אולם הוא יכול לתמוך גם במספר התקנים. התמיכה בהתקני תו רבים נעשית בעזרת טכניקה המאפשרת שימוש במספר כותרות, כאשר לכל התקן יש כותרת אחת. התמיכה בהתקני תו אחדים נעשית על-ידי ציון מספר ההתקנים בשדה שבכותרת ההתקן. במקרה של מספר כותרות, ה-device driver חייב לשרשר אותן באמצעות שדה "הכותרת הבאה", ובכותרת האחרונה להשאיר את השדה הזה ריק.

כותרת ההתקן מאפשרת ל-device driver של התקן התו להוסיף התקן תו חדש לרשימת ההתקנים במערכת, או להחליף התקן קיים שהוכנס על-ידי device driver שהוטען קודם. המפתח להחלפת התקן קיים נמצא בכותרות של שני ה-device driver הטוענים לבעלות על ההתקן. אם שם ההתקן ומאפייניו הנמצאים בכותרת ההתקן החדשה דומים לאלה הנמצאים בכותרת ההתקן הישנה, אז ה-device driver שהוטען קודם מתבקש לוותר על ההתקן שלו, או להתקין אותו מחדש. זאת אומרת שהוא חייב לשחרר את כל המשאבים שהוא הקצה בשמו של ההתקן, כמו פסק בחומרה. רק לאחר שהוא עשה זאת, יוכל ה-device driver החדש לקבל את הבעלות על ההתקן ועליו לתחל אותו.

הטכניקה של החלפת התקני תו קיימים שימושית במיוחד כאשר משפרים את התמיכה בהתקני התו הסטנדרטיים. OS/2 תומכת בצורה אוטומטית בהתקני תו סטנדרטיים כמו, המקלדת (KDB$) והמדפסת (LPT1, LPT2, LPT3). התמיכה בהתקנים אלה נעשית בעזרת קבוצה בסיסית של device drivers המגיעים יחד עם מערכת ההפעלה. device drivers חדשים, או משופרים, יכולים להחליף את אלה המגיעים עם מערכת ההפעלה כדי לתמוך בפונקציות חדשות, או לשפר את הקיימות.

מאידך, device driver התומך בהתקני בלוקים יכול רק להוסיף התקנים לרשימת ההתקנים במערכת. הסיבה לכך היא שאין שום מנגנון שיכול לאפשר ל-device driver זה לציין איזה מבין התקני הבלוקים הוא רוצה להחליף. device driver זה מצביע על מספר ההתקנים שהוא תומך בהם בזמן התיחול שלו, ובהתאם לכך מוקצים לו אותיות כונן. הסדר שבו מופיעים ה-device drivers בקובץ ה-CONFIG.SYS קובע את הסדר שבו יוקצו אותיות הכונן.

התקנים סטנדרטיים ב-OS/2

מספר device drivers מסופקים יחד עם OS/2. ניתן לחלק אותם לשתי קבוצות: אלה שנטענים אוטומטית יחד עם מערכת ההפעלה, ואלה שהמשתמש בוחר בהם על-ידי הצבת המשפט =DEVICE בקובץ ה-CONFIG.SYS. בתרשים 53 תמצא רשימה של device drivers אלה.

שעון

בזמן התיחול, מערכת OS/2 מתקינה בצורה אוטומטית את ה-device driver של השעון. ה-device driver של השעון הוא מסוג המיועד להתקני תו. הוא מנהל את שעון ה-CMOS, שהוא ההתקן המתזמן פונקציות של מערכת הקשורות בזמן.

ה-device driver תומך בק/פ של היישום דרך שירותי השעון הפנימי של OS/2. יישום משתמש בד"כ במישק DosSetDateTime כדי לכתוב להתקן השעון. יישום יכול לקבל את משתני הזמן של המערכת, כמו תאריך/שעה וקצב התקתוק של השעון, על-ידי פניה לסגמנט המידע הגלובלי. יישום יכול גם להשתמש במישקי מערכת הקבצים, כמו DosRead ו-DosWrite, כדי לבצע ק/פ "ישירות" לשעון. כתיבת נתונים לשעון וקריאה ממנו חייבים להיות מאורגנים בקבוצות של ששה בתים.

דיסק/דיסקט

בעת תיחול המערכת, OS/2 מתקינה בצורה אוטומטית את ה-device driver המטפל בדיסק/דיסקט. ה-device driver של הדיסק/דיסקט הוא מסוג המשמש התקני בלוקים ומיועד לנהל את כונני הדיסקטים ואת הדיסקים הקשיחים. לכל אחת מהמחיצות ניתן להכנס עם אות כונן נפרדת. התמיכה בק/פ של היישום נעשית באמצעות מישקי מערכת הקבצים.

מסך

בזמן תיחול המערכת, OS/2 מתקינה בצורה אוטומטית את ה-device driver המטפל במסך. ה-device driver של המסך הוא מסוג המשמש התקני תו. הוא מחליף את החלק של תוכנת device driver של DOS המטפלת בפלט. הוא תומך בק/פ של היישום באמצעות מישקי תת-מערכת VIO.

מקלדת

בזמן התיחול הכללי במערכת, OS/2 מתקינה בצורה אוטומטית את ה-device driver המטפל במקלדת. ה-device driver של המקלדת הינו מסוג המשמש התקני תו. הוא מחליף את החלק המטפל בפלט ב-device driver המתאים של DOS.

ה-device driver של המקלדת תומך בק/פ של היישום דרך מישקי תת-מערכת ה-KBD, ומאפשר גם לתכניות פיקוח (monitor) להתקני תו לבחון את ההקשות, כך שיישום יוכל להשתמש בתכנית פיקוח כדי ליירט את נתוני המקלדת.

מדפסת

בזמן התיחול הכללי במערכת; OS/2 מתקינה בצורה אוטומטית את ה-device driver המטפל במדפסת. ה-device driver של המדפסת הינו מסוג המשמש התקני תו, והוא מנהל את המדפסות המקביליות: LPT1, LPT2 ו-LPT3.

ה-device driver של המדפסת תומך בק/פ באמצעות מישקי מערכת הקבצים ובצורה לא ישירה גם דרך תת-מערכת ה-Print Spool. ה-device driver של המדפסת מאפשר גם לתכניות פיקוח (monitor) של התקני תו לבחון את הפלט הנשלח למדפסת. על כן יישום יכול להשתמש בתכנית הפיקוח הזו כדי ליירט את נתוני ההדפסה.

device driver בסיסיים	device driver ניתנים להתקנה
שעון	עכבר
דיסקט/דיסקט	מחוון הציור
מסך	תקשורת אסינכרונית
מקלדת	דיסק בפועל
מדפסת	דיסקט חיצוני
	ANSI
	EGA

תרשים 53. ה-device drivers הסטנדרטים של OS/2

עכבר

ה-device driver של העכבר מותקן על-ידי המשפט =DEVICE בקובץ CONFIG.SYS. ה-device driver של העכבר הינו מסוג המשמש התקני תו, והוא מנהל את העכבר המקבילי או את העכבר הסידרתי, בהתאם לקביעה בקובץ ה-CONFIG.SYS.

ה-device driver של העכבר תומך בק/פ של היישום באמצעות תת-מערכת ה-MOU. יישום DOS יכול להשתמש במישק DOS Int 33h לצורך ביצוע ק/פ. ה-device driver של העכבר תומך בתכניות פיקוח (monitor) להתקני תו, כך שיישום יוכל להשתמש בתכנית פיקוח כדי ליירט את נתוני העכבר.

מחוון הציור

ה-device driver של מחוון הציור מותקן על-ידי המשפט =DEVICE בקובץ CONFIG.SYS. ה-device driver של מחוון הציור הינו מסוג המשמש התקני תו, והוא מספק מישק ל-device driver של העכבר כדי שזה יוכל לצייר בזמן פסק סמן על המסך. מחוון זה יכול להיות ברירת מחדל, או כזה המסופק על-ידי היישום.

מחוון הציור אינו התקן פיסי לצורך ביצוע ק/פ ולכן הוא אינו תומך בק/פ של היישום.

תקשורת אסינכרונית

ה-device driver של התקשורת האסינכרונית מותקן על-ידי המשפט =DEVICE בקובץ CONFIG.SYS. ה-device driver של התקשורת האסינכרונית הינו מסוג המשמש התקני תו. הוא תומך בהתקנים הסידרתיים COM1 ו-COM2 במחשבי AT ו-XT דגם 286 ובהתקנים הסידרתיים COM1, COM2 ו-COM3 ב-PS/2 דגמים 50, 60, 70 ו-80.

device driver זה מספק ליישומים מישק RS232-C באמצעות IOCtls. השירותים שהוא מספק כוללים: תורי שידור וקליטה, בקרה אוטומטית על אותות הבקרה של המודם ובקרה על זרימת זרם הנתונים לגבי שידור וקליטה. יישום יכול להשתמש גם במישקי מערכת הקבצים, כמו DosRead ו-DosWrite, כדי לבצע ק/פ להתקן הסידרתי. יש להשתמש ב-device driver זה עם תכנית השירות לטיפול בתור למדפסת, אם המדפסת הסידרתית היא זו שאליה מנותב התור.

דיסק בפועל

ה-device driver של הדיסק בפועל מותקן על-ידי המשפט =DEVICE בקובץ CONFIG.SYS. ה-device driver של הדיסק בפועל הינו מסוג המשמש התקני בלוקים והוא מנהל התקן בלוקים המוגדר בזיכרון המערכת. device driver זה תומך בק/פ של היישום דרך מישקי מערכת הקבצים, בדומה להתקן בלוקים רגיל.

דיסק חיצוני

ה-device driver של הדיסק החיצוני מותקן על-ידי המשפט =DEVICE בקובץ CONFIG.SYS, והוא מגדיר התקן לוגי. התקן לוגי זה יכול להיות התקן קיים, ואז מספר אותיות כונן מתייחסות לאותו כונן דיסקטים פיסי. ניתן גם להשתמש בהתקן דיסקים חיצוני למערכת, כמו כונן לדיסקטים 3.5 אינץ'. device driver זה תומך בק/פ של היישום דרך מישקי מערכת הקבצים, כמו כל התקן בלוקים.

ANSI

ה-device driver של ANSI מותקן על-ידי המשפט =DEVICE בקובץ CONFIG.SYS. device driver זה שייך ל-DOS והוא רץ רק בסביבת DOS הפועלת ב-OS/2. הוא מספק מנגנון המרחיב את השליטה במסך ובמקלדת באמצעות הוראות מיוחדות לשליטה על מיקום הסמן, על המאפיינים של התווים המוצגים ועל ההגדרה של המקשים. פונקציות ה-ANSI משולבות בתת-מערכות ה-VIO וה-KBD המוכללות ביישומי OS/2.

EGA

ה-device driver של ה-EGA מותקן על-ידי המשפט =DEVICE בקובץ CONFIG.SYS. device driver זה שייך ל-DOS, ועל כן הוא פועל רק בסביבה זו ב-OS/2. הוא תומך במישק לאוגר EGA ביישומי DOS, והמהווה הרחבה של מישק התצוגה BIOS Int 10h. המישק לאוגר EGA מאפשר ליישומים גרפיים לרוץ בסביבת DOS בצורה כזו, ש-OS/2 יכולה לשמור ולשחזר את המסך בכל פעם שהמפעיל עובר מהיישום הגרפי ואליו. יישום OS/2 יכול גם הוא לבצע פעולות גרפיות ע"י כך שהוא נכנס למאגר תצוגה פיסי עם VIOGetPhysBuf. את המסך הוא יכול לשמור ולשחזר עם VioSaveRedrawWait.

התמיכה של OS/2 ב-DEVICE DRIVERS של DOS

כפי שראית, ניתן להשתמש במספר device drivers של DOS תחת OS/2, עם מספר הגבלות. כדי ש-device driver השייך ל-DOS יוכל לרוץ בסביבת הביצוע של DOS תחת OS/2, הוא חייב למלא אחר הכללים הבאים:

- לטפל בהתקני תווים בלבד.
- לבצע ק/פ בשיטת התיחקור בלבד.
- אסור שתהיה לו תלות בתזמון.
- אסור לו להפעיל אפילו אחת מהקריאות DOS Int 21 בזמן התיחול שלו.

רק יישומי DOS יכולים להשתמש בהתקן המנוהל על-ידי device driver השייך ל-DOS. יישומי OS/2 חדשים אינם רשאים לבצע ק/פ להתקן זה.

ה-device driver השייך ל-DOS מותקן בעזרת מנגנון הזהה לזה שב-DOS, דרך המשפט =DEVICE בקובץ CONFIG.SYS. לאחר שהקובץ הוטען לזיכרון, מתבצע עליו תיחול במצב האמיתי של ה-80286.

פרק **7**

גישות מתקדמות בתכנות

בפרק זה אנו מביאים כמה מבין הגישות היותר מתקדמות בתכנות ב-OS/2, ונדון בפירוט ב**תכנית הקישור של OS/2** (OS/2 Linker) שהוצגה בפרק 4. הדיון מתחיל בדרך שבה יש להשתמש בתכנית הקישור למטרות של יצירת מודולים ברי-ביצוע של היישום (EXE). בדיון זה אנו עוסקים גם באפשרות להגדיר סגמנטים של נתונים ופקודות כנטענים מראש, או כנטענים לפי דרישה. בחלק השני, ניתנת סקירה מעמיקה על המנגנון הקרוי "קישור דינמי" ומוצג השוני בינו לבין מנגנוני הקישור הסטטיים. אנו מתארים את הגישות של קישור דינמי בזמן טעינה ובזמן ריצה לגבי שגרות המוגדרות חיצונית, ומראים איך ניתן להשתמש בתכנית הקישור כדי לבנות ספריות פרטיות המקושרות בצורה דינמית. בחלק השלישי אנו מביאים דוגמת תכנות המציגה כמה מבין האפשרויות שדנו בהן בפרק זה.

קישור של יישום

בפרק 4 הבאנו דוגמה, שבה הפעלנו את תכנית הקישור של OS/2. המקור העיקרי של קלט לתכנית הקישור הינו מודולי-יעד שנוצרו על-ידי מהדר של שפה מסוימת, או שנוצרו בשפת אסמבלר. תכנית הקישור משתמשת במודולים אלה כדי ליצור מודולים של היישום אשר מערכת ההפעלה יכולה לטעון ולבצע. אנו נתייחס להלן למודולים של היישום הניתנים לביצוע **כמודולים של תכניות** (program modules). בהשוואה לסביבת התכנות ב-DOS, סביבת התכנות ב-OS/2 גמישה יותר מבחינת מבנה התכנית וביזור של מרכיבים. תוך כדי תיאור הפונקציות של תכנית הקישור, אנו מכסים רבות מהאפשרויות והבקרות החשובות העומדות לרשות המתכנת כדי לבנות מודול של תכנית תוך שימוש בתכונות הגמישות של OS/2.

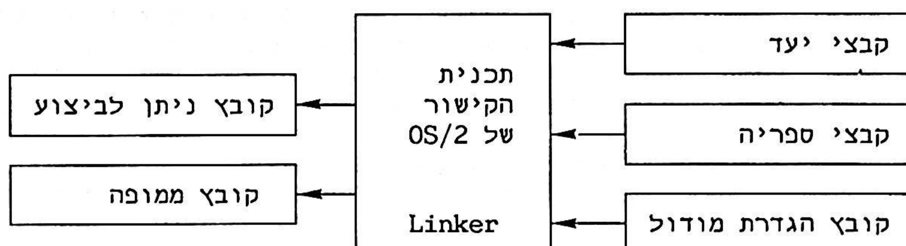
הבה נסתכל תחילה בתחביר של שורת הפקודה המפעילה את תכנית הקישור (ראה תרשים 54). אנו רוצים לכסות נושאים נוספים שלא הובאו בדוגמה בפרק 4.

```
LINK object files, [executable file], [map file],
        [library files], [module definition file]
                [/option] [/option]
```

— — — — — — — — —

```
LINK קבצי יעד, [קובץ ניתן לביצוע], [קובץ מיפוי],
        [קבצי ספריה], [קובץ הגדרת מודול],
                [/אופציה] [/אופציה]
```

תרשים 54. תכנית הקישור - התחביר של שורת הפקודה



תרשים 55. קבצי הקלט והפלט של תכנית הקישור

דנו כבר בפרמטרים של תכנית הקישור העוסקים בקבצי יעד, קבצים ברי-ביצוע וקבצי ספריה. נקודה חשובה מאוד שאינה מצוינת בדוגמה הקודמת היא, שניתן לרשום יותר מקובץ־יעד אחד בפרמטר הראשון. תוספת זו מאפשרת לקשר מספר מודולי-יעד למודול יחיד הניתן לביצוע. ניתן להבין מכך, שניתן לא רק לפתח תכנית בצורה מודולרית, אלא שניתן גם להרכיב תכנית שחלקיה כתובים בשפות שונות. לדוגמה, החלק הגדול של היישום יכול להכתב בשפת C, ואת השגרות הרגישות לזמן ביצוע אפשר לכתוב בשפת Assembly.

קובץ הגדרת מודול הינו מאפיין של תכנית הקישור של OS/2, אשר מכיל מידע שמשתתף בתהליך הקישור. הוא מספק קלט לתכנית הקישור בנוסף לקלט המתקבל מהפרמטרים בשורת הפקודה. קובץ זה אינו חובה כאשר מבצעים קישור של מודול תכנית. הוא מאפשר למתכנת לעקוף את ברירות המחדל של תכנית הקישור לגבי מאפיינים מסוימים של סגמנטים (כמו גודל הסגמנט במחסנית). בתרשים 55 תמצא תרשים סכימטי של הקבצים המשמשים כקלט לתכנית הקישור.

אנו משיגים שני דברים עיקריים בעזרת תהליך הקישור. הראשון, תכנית הקישור אחראית להמרת הכתובות בהפניות החיצוניות הנמצאות במודול היעד. והשני, תכנית הקישור חייבת להכניס מידע מתאים לתוך הקובץ הניתן לביצוע שיאפשר למטען (loader) לטעון את התכנית לזיכרון. בסעיף הדן בקישור דינמי נראה, שב-OS/2 המרת הכתובות בהפניות החיצוניות לשגרות הקישור הדינמי אינה מלאה עד אשר התכנית נטענת למעשה. אבל לפני שנתחיל בקישור דינמי, הבה ונראה את התכונה של OS/2 להגדיר סגמנטים נטענים מראש (Preload), וסגמנטים נטענים לפי דרישה (Load on Demand, או Load on Call).

סגמנטים הנטענים מראש

כאשר תכנית DOS מוטענת, כל הסגמנטים שלה מובאים לזיכרון. המטען (loader) משנה את הכתובות היחסיות בתכנית (שחושבו על-ידי המהדר ועל-ידי תכנית הקישור), לכתובות זיכרון שבהן היא תשתמש בעת הביצוע. תהליך זה מסתיים לפני שהשליטה עוברת לנקודת הכניסה הראשית של התכנית. סגמנטים של OS/2 שמוטענים לפני שהתכנית מקבלת את השליטה נקראים **סגמנטים נטענים מראש** (preloaded segments). ניתן להשתמש בשלושה סוגי משפטים בקובץ להגדרת המודול כדי לציין לתכנית הקישור שיש צורך לטעון את הסגמנטים מראש. הם גורמים לתכנית הקישור לבנות את המידע הדרוש במודול הסופי שניתן לביצוע, כדי להורות למטען לטעון את הסגמנטים של הנתונים והפקודות מראש.

ניתן להשתמש במשפטי CODE ו-DATA כדי להצביע על כך שיש לטעון מראש, בהתאמה, את כל הסגמנטים של הפקודות, או את כל הסגמנטים של הנתונים, הנמצאים במודול הטעינה. לדוגמה:

```
CODE PRELOAD
```

משפט זה מצביע על כך שיש צורך לטעון מראש את כל הסגמנטים של הפקודות במודול זה.

כדי לציין מאפייני טעינה שונים לסגמנטים שונים במודול התכנית, יש צורך להשתמש במשפט SEGMENTS. לדוגמה:

```
SEGMENTS
    CODESEG1 PRELOAD
    CODESEG2 LOADONCALL
```

ועכשיו, לאחר שהבנו את המשמעות של סגמנט הנטען מראש, הבה נדון במשמעות של "LOADONCALL", או **טעינה לפי דרישה.**

סגמנטים הנטענים לפי דרישה

יישומים הופכים להיות מורכבים יותר ויותר, ולכן ישנה נטיה לכלול ביישום לוגיקה שייתכן ויהיה בה צורך בתנאים מסוימים, אך אין בה צורך בפעילות יומיומית של היישום. במערכת רבת משימות מאוד לא רצוי שהפקודות, שהן התוצאה של הלוגיקה הנ"ל, יהיו במערכת. הסיבה לכך היא שכאשר לא משתמשים בהן (וזה רוב הזמן) הן תופסות חלקים ממרחב הכתובות האמיתי, או ממרחב הכתובות בפועל של התכנית. ככל שהמערכת הופכת להיות עמוסה יותר, אין אנו יכולים להרשות לעצמנו לבזבז את מרחב הכתובות של קובץ הסגמנטים המועברים. מסיבה זו, OS/2 מאפשרת להגדיר סגמנטים כפי שראינו בדוגמה הקודמת. LOADONCALL פירושו, שהמטען שומר ערך בורר לסגמנט זה בטבלת ה-LDT של היישום, אבל הוא אינו טוען למעשה את הסגמנט עד אשר היישום צריך אותו. הצורך בסגמנט זה מתעורר כאשר מטעינים את ערך הבורר שלו לתוך אוגר הסגמנט. רק כאשר זה קורה, המטען מביא את הסגמנט מהדיסק אל הזיכרון ומסיים לערוך את מתאר הסגמנט שלו.

כדי לשמור על משאבי מערכת ב-OS/2, ברירת המחדל לטעינת סגמנטים היא LOADONCALL, הן לגבי סגמנטים מכילי פקודות והן לגבי סגמנטים מכילי נתונים.

קישור דינמי

קישור דינמי הינו אחד המרכיבים הבסיסיים של OS/2, אשר תורם לשיפורים רבים במאפיינים התפעוליים של המערכת. קודם הזכרנו שתכנית הקישור אחראית להמרת הכתובות בהפניות החיצוניות שבמודול היעד. **קישור דינמי** (dynamic linking) הוא מנגנון שמעכב את המרת הכתובות בהפניות חיצוניות אלה, עד לאחר תהליך הקישור ולכן הוא נקרא לפעמים **קישור מעוכב** (delayed binding). למעשה, תכנית הקישור מציבה מידע אודות כל הפניה חיצונית לקישור דינמי בתוך מודול התכנית, ובכך היא מאפשרת למטען המיוחד של הקישור הדינמי להמיר את הכתובות בהפניות החיצוניות בזמן מאוחר יותר. הבה נסתכל על שני הסוגים של קישור דינמי ב-OS/2.

קישור דינמי בזמן טעינה

כפי שתצפה בוודאי, הכתובות בהפניות לשגרות המקושרות באופן דינמי להלן, שגרות דינמיות, מוסבות לכתובות אחרות, כאשר מודול התכנית נטען לזיכרון. זו דרך אופיינית לטפל בכתובות ההפניה לספריות של שגרות דינמיות. כפי שציינו בהקדמה יש מספר יתרונות למישק **קישור דינמי בזמן טעינה** (load time dynamic linking).

אחד היתרונות של מנגנון הקריאה הוא בכך, שמאותו רגע שבו התבצעה המרה של כתובת ההפניה, יש נקודת כניסה ישירה לתוך הפונקציה לכל מי שרוצה בכך. ב-DOS, הגישה לרוב הפונקציות של מערכת ההפעלה מתבצעת על-ידי הצבה של מספר הפונקציה וכל הפרמטרים באוגרים והפעלה של פסק תוכנה 21. סוג זה של מנגנון דורש שפונקציית ניתוב תתבצע בין בקשת היישום לבין נקודת הכניסה התחילית של הפונקציה הרצויה. כאשר משתמשים במישק הקישור הדינמי של OS/2, אין צורך בפונקציית ניתוב זו. כאשר מתבצעת טעינה של המודול שסוגו EXE, המחוונים של ההפניות החיצוניות מצביעים ישירות על נקודות הכניסה של הפונקציות הרצויות ב-OS/2. ביטול הצורך בפונקציית ניתוב במנגנון הקישור הדינמי, מהווה שיפור בביצועים של המערכת.

יתרון נוסף של מישק הקישור הדינמי הוא בכך, ששפות עיליות יכולות להשתמש בו באופן ישיר. המהדר יכול להכנס ישירות לפונקציות של מערכת ההפעלה על-ידי דחיפת פרמטרים לתוך מחסנית וביצוע קריאה רחוקה. פעולה זו זהה לפעולה שבה שפת תכנות משתמשת כדי לקרוא לתת-שגרות בתוך היישום. ב-DOS, קישורים בשפות תכנות משמשים כדי להוציא פרמטרים מהמחסנית, לטעון אוגרים ולהפעיל פסקי תוכנה. אין צורך בקישורים אלה בשפות תכנות של OS/2.

קל יותר להרחיב את התפקודיות של מישק תכנות המבוסס על מנגנון הקישור הדינמי. הוספה של פונקציה למישק שיש לו נקודת כניסה יחידה ופונקציית ניתוב, דורש שינוי בפונקציית הניתוב. מאידך, ניתן להוסיף פונקציות המקושרות בצורה דינמית למערכת מבלי לפגוע בפונקציות מגירסה קודמת, ואין כל פונקציית ניתוב שצריך לעדכן אותה. יתר על כן, המחסנית הינה מנגנון גמיש ועקבי יותר להעברת פרמטרים מאשר האוגרים השונים של המיקרו-מעבד הנתמכים על-ידי מערכת ההפעלה.

קישור דינמי הופך את התמיכה בספריות של שגרות לקלה יותר. שגרות ספריה שמקושרות דינמית ניתנות לשינוי ועדכון מבלי לפגוע ביישומים שמשתמשים בהן. לעומת זה, תכניות שמשתמשות בשגרות ספריה המקושרות בצורה סטטית, חייבות לעבור קישור מחדש כאשר יש צורך להשתמש בגירסה חדשה של השיגרה. בקישור דינמי, כל עוד מספר הפרמטרים לא השתנה, היישום הישן יוכל להשתמש בגירסה החדשה של פונקציית הספריה מבלי לקשור אותה מחדש. בגירסה החדשה יכולה להיות קבוצה חדשה של שגרות דינמיות התומכות ביישומים הקיימים בצורה תואמת לזו של השגרות הישנות. שגרות אלו גם יכולות לתמוך בפונקציות חדשות הנובעות מטווח מורחב של ערכים השייך לפרמטר אחד או יותר שהוגדר קודם. הדבר מקל על שיפור שגרות ספריה כדי שיתאימו לדרישות המשתמש.

ולבסוף, במישק לתכנות יישומים המקושר דינמית נמצא את כל היתרונות של ארכיטקטורה פתוחה. הארכיטקטורה הפתוחה ב-OS/2 מאפשרת ליישומים לקרוא לשגרות המקושרות דינמית שהוספו על-ידי מפתחי תוכנה עצמאיים באותה דרך שהם קוראים לשגרות שסופקו על-ידי מערכת ההפעלה. כאשר בידי מפתחי תוכנה

עצמאיים יש פונקציות חדשות, הם יכולים לשווק אותן למשתמשי OS/2 כמודולים לקישור דינמי, ובכך להרחיב את האפשרויות והתכונות של מערכת ההפעלה.

קישור דינמי בזמן ריצה

קישור דינמי בזמן ריצה (run time dynamic linking) הוא מנגנון, אשר מאפשר לדחות את הקשירה של הפניות חיצוניות במודול התכנית עד לאחר הטעינה. דוגמאות לתועלת שמביא מנגנון זה כוללות כל מצב, שבו יישום רוצה לבצע עיבוד מסוים לפני שהוא מציין את שם המודול שהוא רוצה בו. לדוגמה, יישום יכול לאפשר למשתמש, לבחור בין שיפור ביצועים לבין שיקולי ניצול זיכרון, לפני שהוא מפעיל פונקציה מסוימת. דוגמה אחרת יכולה להיות בחירה של שגרות בהתאם למאפיינים מסוימים של תצורת המערכת. כך ניתן להשתמש בדרכים שונות עם אותה קבוצה של קריאות לפונקציות מתוך יישום יחיד.

התמיכה של OS/2 בקישור דינמי בזמן ריצה נעשית באמצעות קבוצה מיוחדת של פונקציות. DosLoadModule טוען מודול מקושר דינמית. DosFreeModule משחרר מודול שהוטען קודם. DosGetProcAddr משיג את הכתובת של פרוצדורה המקושרת בצורה דינמית. DosGetModHandle משיג מקשר למודול המקושר בצורה דינמית (מודול דינמי). DosGetModName מקבל את שם המודול הדינמי. פונקציות אלו מספקות מנגנונים המאפשרים לבצע בצורה דינמית איתור, טעינה וקישור של מודולים חיצוניים.

כדי שתוכל להעריך את הגמישות בהפעלה המושגת באמצעות פונקציות אלו, הבה נבחן אותן מקרוב.

DosLoadModule טוען מודול המקושר דינמית (להלן, מודול דינמי) לתוך הזיכרון. המודול הזה ניתן לביצוע (קובץ .EXE) ויש בו נקודת כניסה אחת או יותר לפרוצדורות המקושרות דינמית. היישום מעביר למודול הדינמי מחוון למחרוזת ASCIIZ המכילה את שם הקובץ. מסלול החיפוש בספריה משמש לאיתור המודול הדינמי במערכת הקבצים. אם הפונקציה מצליחה לטעון בהצלחה את המודול, היא מחזירה מקשר. ומכאן ואילך, שאר הפונקציות של OS/2 התומכות בקישור דינמי בזמן ריצה, משתמשות במקשר למודול ולא בשמו. אם לא ניתן לטעון את המודול הדינמי, משתמשים בשני פרמטרים אחרים של הפונקציה, כדי להחזיר את שם האובייקט שגרם לכשלון.

DosFreeModule מודיע למערכת ההפעלה שהתהליך סיים להשתמש בהפניות חיצוניות לפרוצדורות בתוך המודול הדינמי. אם אף לא תהליך אחד במערכת משתמש במודול, הסגמנטים שלו ימחקו מהזיכרון. המקשר למודול משמש במקרה זה להצבעה על המודול הדינמי שיש לשחרר.

DosGetProcAddr מחזיר כתובת שניתן להשתמש בה לקריאה לפרוצדורה בתוך המודול הדינמי. התהליך שקרא לפונקציה זו מעביר לה כפרמטרים את המקשר למודול ומחרוזת עם שם הפרוצדורה. ניתן להשתמש במנגנון אלטרנטיבי אחר כדי לציין את הפרוצדורה בתוך המודול, ובמקום להעביר את השם ניתן להעביר את מספר הפרוצדורה במודול. מספר זה הוא המספר הסידורי של הפרוצדורה בתוך המודול.

DosGetModHandle בודק אם המודול הדינמי הוטען כבר לזיכרון. התהליך

שקרא לפונקציה זו מעביר מחוון למחרוזת ASCIIZ המכילה את שם המודול. אם המודול הוטען כבר לזיכרון, מוחזר מקשר למודול. אם המודול עדיין לא הוטען לזיכרון, מוחזרת הודעת שגיאה.

DosGetModName מאתר את המודול הדינמי בתוך מערכת הקבצים. לפונקציה זו מועברים כפרמטרים מקשר למודול ומחוון למאגר. שם הקובץ המלא הכולל את הכונן, המסלול, שם הקובץ וההרחבה, מוחזר למאגר.

יצירה של ספריית קישור דינמי

נחזור בקצרה על מה שלמדנו עד עתה: קישור דינמי בזמן טעינה ובזמן ריצה. קישור דינמי בזמן טעינה מציין מצב שבו שגרות דינמיות יוטענו בעת שתכנית היישום מוטענת. בזמן הטעינה, מטען הקישור מציב ערכים בכל הכתובות במודול התכנית, כדי לאפשר קשר לשגרות ספריה דינמיות. קישור דינמי בזמן ריצה מציין מצב, שבו שגרות דינמיות נטענות ומקושרות לפי דרישה של היישום, המשתמש לצורך זה בקבוצה מיוחדת של פונקציות OS/2. מנגנוני הקישור הדינמיים חיוניים באותה מידה הן בזמן טעינה והן בזמן ריצה. ספריות הקישור הדינמי לזמן טעינה ולזמן ריצה נבנות באותה דרך.

MYPROGRAM.EXE
(מודול תכנית)

כותרת של קובץ EXE . . .
MYLIB.DLL
שדה כתובת 1
שדה כתובת 2
. . .
פקודות בשפת מכונה ונתונים . . .

תרשים 56. קובץ המכיל מודול תכנית

תכנית הקישור של OS/2 בונה מודול תכנית עם מבני נתונים ההכרחיים לתמיכה בקישור דינמי. מבני הנתונים החשובים שנדון בהם, הם שם קובץ של מודול דינמי, ושם הפונקציה (או המספר הסידורי שלה) המצביעה על פונקציה מסוימת המקושרת בצורה דינמית. תרשים סכימה של מודול התכנית המכיל את מבני הנתונים הללו נמצא בתרשים 56.

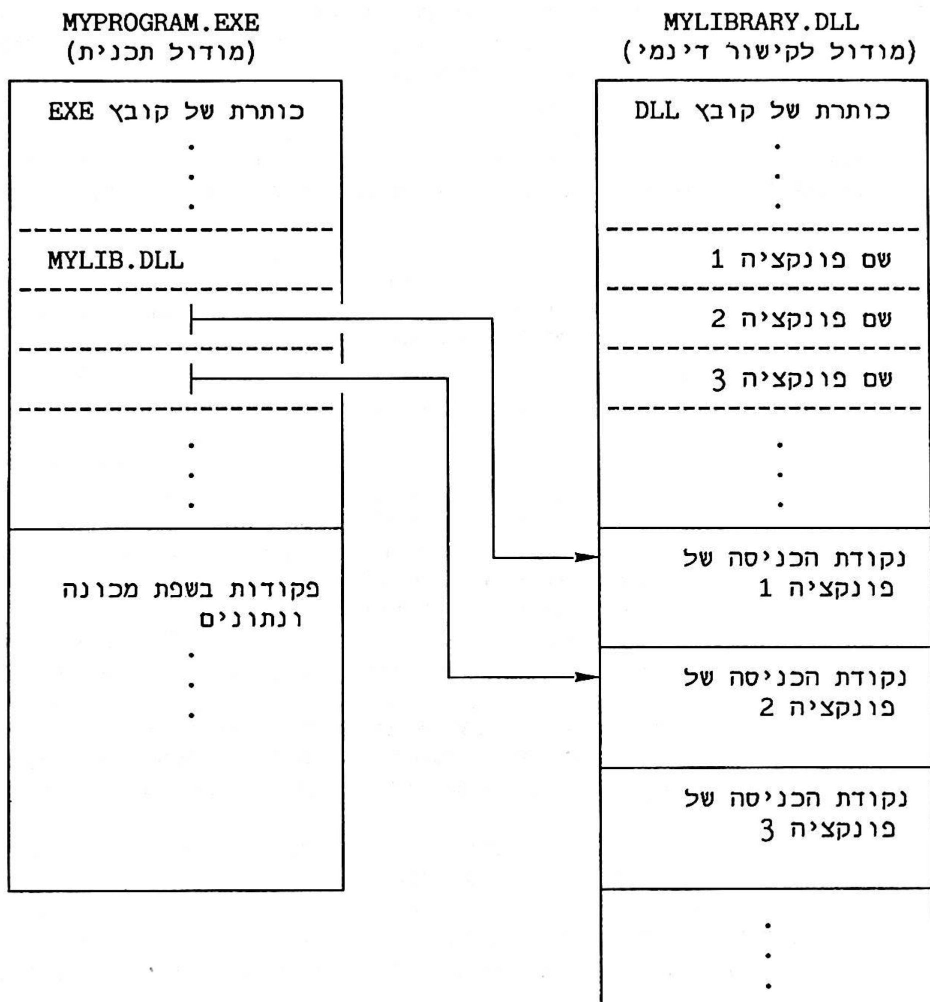
תכנית הקישור של OS/2 יכולה לבנות מודול דינמי שמכיל את הספריה של השגרות שהקישור לגביהן מעוכב עד זמן טעינה, או עד זמן ריצה. אחת הדרכים לתאר מודול המקושר דינמית מוצגת בתרשים 57.

כעת, אם נסתכל במודולים אלה לאחר שהם הוטענו לזיכרון ונוצר הקשר ביניהם, נגיע לדיאגרמה הנראית בתרשים 58.

MYLIBRARY.DLL
(מודול מקושר דינמית)

כותרת של קובץ DLL . . . --- שם פונקציה 1 --- שם פונקציה 2 --- שם פונקציה 3 --- . . .
נקודת הכניסה של פונקציה 1
נקודת הכניסה של פונקציה 2
נקודת הכניסה של פונקציה 3
. . .

תרשים 57. קובץ המכיל מודול המקושר דינמית



תרשים 58. התוצאות בזיכרון של קישור דינמי

```
;
; Module Definition File
; Created by JIK, AMM, and RLW
; 1987
;
LIBRARY
DESCRIPTION 'MYLIBRARY Ver 1.0 Copyright 1987 JIK, AMN, RLW'
EXPORTS
    Function1   @1
    Function2   @2
    Function3   @3
```

תרשים 59. קובץ הגדרה של מודול לקישור דינמי

אנו רואים ששדות הכתובות שתכנית הקישור בנתה במודול התכנית מתמלאים עם מחוונים. בעזרת מחוונים אלה ניתן לפנות ישירות לנקודות הכניסה של הפונקציות המתאימות להם. ועכשיו, לאחר שמנגנון הקישור הדינמי סידר את התאמת הכתובות, ניתן לקרוא לפונקציה הנמצאת בספריה המקושרת בצורה דינמית כאילו היתה תת-שיגרה שסופקה על-ידי התכנית המקורית. אם נשתמש במינוח אחר, נוכל לומר שתכנית היישום מייבאת בצורה יעילה פונקציות מספריה המקושרת בצורה דינמית ומשתמשת בהם כאילו היו חלק של התכנית המקורית.

תכנית שירות המגיעה עם OS/2 בונה קובץ מיוחד המשמש את תכנית הקישור לבניית שדות הכתובות במודול התכנית עבור פונקציות מיובאות. קובץ מיוחד זה נקרא ספריית יבוא, ותכנית השירות היוצרת את הספריה נקראת ספרנית היבוא, או תכנית השירות IMPLIB.

ספרנית היבוא

ספרנית היבוא (import librarian - IMPLIB) משמשת ליצירת קבצי ספריה מיוחדים לשגרות דינמיות, בתהליך נפרד מתהליך הקישור. ספריות של שגרות המקושרות בצורה סטטית מכילות את קוד היעד לשגרות אלו, אשר מועתקות למודול התכנית בזמן הקישור. כדי לתמוך בשגרות המקושרות בצורה דינמית, ספרנית היבוא יוצרת קבצי ספריה מיוחדים המאפשרים לתכנית הקישור לבנות בתוך מודול התכנית את השדות המיועדים לאחזקת הכתובות.

ספרנית היבוא בונה ספריית יבוא על-ידי עיבוד קובץ הגדרת המודול. בתרשים 59 תמצא דוגמה של קובץ הגדרת מודול, המשמש לקישור מודול דינמי. אותו קובץ משמש גם להפקת ספריית היבוא המכילה את השגרות הדינמיות.

תכניתן היישומים יכול לציין באופן ידני מודולים של פונקציות מיובאות שיחליפו את ההפניות החיצוניות. גם תכנית הקישור יכולה להשתמש בספריית היבוא כדי להחליף בצורה אוטומטית את ההפניות החיצוניות האלו. ספריות יבוא מאפשרות למתכנת לפתח תכניות המשתמשות בשגרות דינמיות באותה קלות של פיתוח תכניות המשתמשות בשגרות המקושרות בצורה סטטית.

בדוגמה זו, אנו רואים שני משפטים המאפיינים קובץ זה כקובץ המגדיר מודול לקישור דינמי. המשפט LIBRARY מכריז שהקובץ להגדרת המודול מתייחס לספריה של שגרות דינמיות. המשפט EXPORTS מאפשר להגדיר:

- שמות לנקודות הכניסה.
- מספרים סידוריים לנקודות הכניסה.
- כל מאפיין הקשור לשגרות שמעמיד לרשותנו מודול זה המקושר בצורה דינמית.

אנו רואים, שהקובץ להגדרת המודול שמשמש את תכנית הקישור ליצירת מודול המקושר דינמית מסייע גם לספרנית היבוא ליצירת קבצים דינמיים בספריית היבוא. קבצי ספריית היבוא מקלים על המתכנת לקשור יישומים שיש להם הפניות חיצוניות לשגרות המקושרות דינמיות.

משפטי הקובץ להגדרת המודול

כדי לקבל תמונה מלאה יותר של האופציות העומדות לרשותך, כדי ליצור מודולים של תכניות ומודולים המקושרים דינמית, נסכם את המשפטים שניתן להשתמש בהם בקובץ להגדרת המודול.

NAME מציין שהמודול הניתן לביצוע הוא מודול תכנית. ניתן לציין במשפט זה שם למודול התכנית. OS/2 משתמשת בשם כדי לזהות את מודול התכנית לאחר שהוא הוטען. אם לא צוין שם המודול, מערכת ההפעלה משתמשת בשם הקובץ של המודול כדי לזהות אותו.

LIBRARY מציין שהמודול שנוצר הוא מודול ספריה דינמי. ניתן לציין שם ספריה כדי לעקוף את ברירת המחדל המשמשת את OS/2 בזיהוי המודול, לאחר שהוא הוטען לזיכרון.

בספריות דינמיות שנדרשת בהן שיגרת תיחול, ניתן לציין פרמטר נוסף במשפט LIBRARY הגורם לתיחול הספריה. משפט LIBRARY יכול לציין שיש לקרוא לשיגרה זו, המתחלת את הספריה, רק פעם אחת כאשר הספריה נטענת לראשונה, או שיש לקרוא לה בכל פעם שתהליך חדש מצליח להכנס לספריה.

DESCRIPTION מאפשר להכניס שורה יחידה של טקסט לתוך מודול התכנית. משפט זה מקל על ההתאמה בין מודול התכנית לבין הגירסה של קובץ תכנית המקור. ניתן להשתמש במשפט זה להכנסת משפט סטנדרטי המצהיר על זכויות יוצרים.

CODE מגדיר את ברירת המחדל למאפיינים של כל הסגמנטים המכילים פקודות בתוך מודול. ניתן לציין את אופציות הטעינה "נטען מראש - Preload" או "נטען לפי דרישה - LoadOnCall". ניתן גם לקבוע זכויות גישה לסגמנטים: ביצוע בלבד (execute only), או ביצוע עם קריאה (execute read). ניתן גם לציין פרמטר המצביע על האפשרות שהסגמנטים של הפקודות יקבלו הרשאת ק/פ. הרשאה זו מאפשרת לשגרות בסגמנט להשתמש בפקודות המוגנות של ה-80286 כולל הוראות IN ו-OUT.

DATA מגדיר את ברירת המחדל למאפיינים של כל הסגמנטים המכילים נתונים בתוך מודול. ניתן לציין את אופציות הטעינה "נטען מראש - Preload", או "נטען לפי דרישה - LoadOnCall". ניתן גם לקבוע זכויות גישה לסגמנטים: קריאה בלבד (read only), או קריאה/כתיבה (read/write).

פרמטר המציין את השימוש בסגמנט הנתונים האוטומטי של המודול, מוקצה לנתונים בצורה אוטומטית מבלי שהיישום נקט בפעולה כלשהי. משפט DATA יכול לציין שלא קיים סגמנט נתונים אוטומטי. מודולים המקושרים דינמית יכולים להשתמש בפרמטר זה כדי לציין שסגמנט הנתונים האוטומטי קיים ויש להתחלק בו (הכוונה לתהליכים) בכל מצב שבו נמצא המודול המקושר דינמית. ניתן גם להשתמש בפרמטר זה כדי לציין שיש צורך ליצור סגמנט נתונים אוטומטי בכל פעם שמודול זה נטען לזיכרון.

פרמטר נפרד מציין שיתוף של סגמנטים אחרים מכילי נתונים במודול. שיתוף זה מכוון לכך שכל התהליכים הנכנסים למודול יכולים להשתמש בעותק אחד של הסגמנטים מכילי הנתונים. סימון סגמנטים אלה כ"לא ניתנים לשיתוף" גורם ליצירת עותק נפרד של הסגמנטים המוגדרים לקריאה ו/או כתיבה לכל תהליך

המשתמש במודול זה. למשפט DATA יש פרמטר המציין אם למקם את הסגמנטים מכילי הנתונים ברמת הרשאת ק/פ.

SEGMENT מאפשר ציון נפרד של מאפיינים לכל סגמנט נתונים או סגמנט פקודות. ניתן להשתמש באותם פרמטרים העומדים לרשות המשפטים CODE ו-DATA, ביחד עם פרמטר שם הסגמנט, המצביע על הסגמנט שמעניין אותנו.

STACKSIZE מציין את מספר הבתים שיש להקצות למחסנית הקשורה למודול התכנית. ערך זה מבטל כל מחסנית שצוינה בתכנית המקורית.

IMPORTS מספק דרך נוספת להמרת הפניות חיצוניות לשגרות ספריה דינמיות. באופן נורמלי, הפניות אלה יומרו על-ידי ספריית היבוא. אם לא ניתן להשתמש בספריית היבוא, המתכנת יכול לציין את נקודות הכניסה עם המשפט IMPORTS.

פרמטרים למשפט זה כוללים:
- שם שניתן להשתמש בו בתכנית לשם פנייה לשיגרה דינמית.
- שם המודול לקישור דינמי שמכיל את השיגרה.
- שם הפונקציה המבוקשת, או המספר הסידורי שלה בתוך המודול.

EXPORTS מציין את כל נקודות הכניסה לתוך מודול דינמי, כדי לאפשר קריאה לפונקציה ממודול אחר. כאשר מתייחסים לפונקציה מתוך המודול עצמו, יש לה שם נוסף שניתן להשתמש בו במקום השם שמשמש ליצוא.

ניתן להשתמש גם בפרמטר המציין את המקום של הפונקציה בטבלה של נקודות הכניסה המתייחסת למודול. הדבר מאפשר להפעיל את הפונקציה בעזרת השם המשמש ליצוא, או בעזרת המספר הסידורי.

ניתן להשתמש בפרמטר נוסף לגבי פונקציות הנקראות בד"כ בשמן, גם אם משתמשים באופציה של המספר הסידורי. פרמטר זה מכוון את מערכת ההפעלה לשמור בזיכרון את המחרוזת המכילה את שם הפונקציה, ובכך להשיג ביצוע אופטימלי בזמן הפנייה לשם הפרוצדורה על-פי ההתייחסות אליה.

HEAPSIZE מציין את מספר הבתים שמודול התכנית צריך לשימוש עצמי.

PROTMODE מציין שיש לבצע את המודול הזה רק במצב מוגן. משפט זה יוצר מציין במודול, אשר גורם לתכנית הקישור להשמיט מידע מהמודול שיכול לשמש את תכנית השירות BIND בעת בניית יישומי משפחה. PROTMODE שומר מרחב כתובות במודולים שאינם מיועדים להכנס ליישומי משפחה.

OLD עוזר לשמור על העקביות בהקצאה של מספרים סידוריים לפונקציות בכל הגירסאות שבספריה המקושרת בצורה דינמית. אם משתמשים במשפט זה, מתבצע חיפוש ברשימת השמות של נקודות הכניסה במודול החדש כדי לבדוק את התאמתן לאלה הנמצאות במודול הישן שצוין. אם נמצאה התאמה, מוסיפים כניסה בטבלת נקודות הכניסה עבור המודול החדש, ונותנים לפונקציה מספר סידורי הזהה לזה שהיה לה במודול הישן.

תכנית דוגמה לקישור דינמי בזמן ריצה

בדוגמה זו, מוצגים כמה מהכלים של העורך (linker) המיועדים לבניית ספריה המקושרת בצורה דינמית. הדוגמה מתבססת על התכנית העוסקת במספר תהליכים, בכך שהפכנו את תהליך 2 לשיגרה המקושרת בצורה דינמית. לאחר מכן השתמשנו בקישור דינמי בעת ריצה כדי לפנות לאחת משתי הגירסאות של מודול זה.

תהליך 2 בתכנית הדוגמה, העוסקת במספר תהליכים, מבצע הדמייה של תהליך לרישום של אירוע עם התאריך והזמן. נניח שהתבקשנו לבנות את תהליך הרישום הזה בצורה כללית, כך שניתן לשנות בקלות את צורת הרישום של התאריך והזמן. אנו משתמשים באמצעים העומדים לרשות הקישור הדינמי ב-OS/2 כדי לפתור בעיה זו. על-ידי הפרדה של הפונקציות שבונות את התאריך והזמן משאר התכנית, לפנינו משימה פשוטה כדי לספק מספר גירסאות לפונקציות אלה, אשר עורכות את התאריך והזמן בצורה שונה. כדי לפשט את הדוגמה שלנו, אנו בונים רק שתי גירסאות של ספריות אלו, ונותנים להן את השמות "USA Format" ו-"European Format".

הבה נסתכל בשינויים הספציפיים בתהליך 1 ובתהליך 2 של הדוגמה העוסקת במספר תהליכים. השלב הראשון בבניית הדוגמה מחייב להוסיף לתהליך 1 שיגרת ק/פ שתהיה בשליטת המשתמש. שיגרה זו נקראת SelectLibrary. היא פותחת חלון במסך ומבקשת מהמשתמש לבחור את המערך הרצוי של התאריך והזמן. מכיון שיש לנו שתי אפשרויות בלבד, שיגרה זו מציגה רק אותן, ומאפשרת למשתמש לבחור באחת מהן בעזרת מקשי החצים. זכור, שניתן לתמוך במספר בלתי מוגבל של מערכים על-ידי כך שניתן למשתמש להכניס את שם הספריה המסוים. זו רק דוגמה אחת של העוצמה והגמישות של מנגנון הקישור הדינמי.

הצבנו את השיגרה SelectLibrary בתהליך 1, כי הוא אחראי לקבלת הקלט מהמשתמש באמצעות המקלדת. תהליך 2 הוא למעשה שיגרה המשתמשת בתוצאות של בחירת המשתמש, כדי להפעיל את הספריה הדינמית המתאימה, ולכן אנו חייבים להעביר אליו את המידע הזה כמשתנה, בזמן שהוא נוצר.

השינויים בתהליך 2 מתחילים בהוצאת הפונקציות BuildDateString ו-BuildTimeString מהתכנית. את הפונקציות האלו הפכנו למודולים של ספריה המקושרים בצורה דינמית. רמז אחד שיכול להקל על ההמרה הזו הוא להדר ולקשר את המודולים האלה בעזרת מודל התכנות הרחב בשפת C. אנו משתמשים במודל זה כאשר לתכנית יש סגמנטים רבים של פקודות וסגמנטים רבים של נתונים. לכן, המהדר משתמש בברירת מחדל המאפשרת לו לבצע מיעוו בין סגמנטים, מכיון שהוא מניח שכל מחוון, קריאה לפונקציה וחזרה ממנה עלולים לחרוג מתחום הסגמנטים. שגרות המקושרות בצורה דינמית נטענות למעשה לזיכרון בסגמנטים השונים מאלה של התכנית שקראה להן, וכתוצאה מכך הנושא של מיעון בין סגמנטים הופך להיות חשוב ביותר. כאשר תסתכל בתשומת לב בדוגמת התכנית, תוכל לראות מספר דוגמאות שבהן נקטנו בפעולות מסוימות, כדי לענות על הדרישות של תכניות עם סגמנטים רבים.

יתרת השינויים בתהליך 2 כוללים את ההכרזות, האלגוריתמים ואת הפונקציות הדרושות כדי להפעיל קישור דינמי בזמן ריצה.

```
הפונקציה SelectLibrary
  הצג את חלון הקלט למשתמש
  האר את הספריה הנבחרת
  בצע עד אשר ילחץ מקש Enter
     קרא מהמקלדת
     אם לחצת על אחד ממקשי החצים מעלה-מטה,
        אז, אם הספריה הנבחרת כרגע היא 1,
            אז       בחר את ספריה 2
                     הסר הארה מספריה 1
                     האר ספריה 2
            אחרת,   בחר את ספריה 1
                     הסר הארה מספריה 2
                     האר ספריה 1
        אחרת, אם נלחץ מקש Enter,
            אז      אל תעשה דבר
            אחרת,  השמע קול כי נלחץ מקש
                     לא נכון
  מחק את חלון הקלט למשתמש
  חזור
```

תרשים 60. הלוגיקה של הפונקציה SelectLibrary

האפשרות של המשתמש לבחור ספריה

כדי לבנות דוגמה זו, הכנו ארבעה קבצי מקור וקובץ להגדרת המודולים של הספריה. ארבעת קבצי המקור הם תהליך 1, תהליך 2, ושתי הגירסאות של הספריות הדינמיות לפונקציות העורכות את הזמן והתאריך. במקום להעתיק את כל התיאור והתכנית של תהליך 1, אנו מראים רק את ההכרזות והאלגוריתמים הנוספים שיש צורך להוסיף לפונקציה SelectLibrary. נסתכל תחילה בלוגיקה של SelectLibrary, אשר מוצגת בתרשים 60.

כמו בכל הדוגמאות, הוספנו את הכרזות המשתנים לתחילת קובץ המקור. הקריאה לפונקציה SelectLibrary נמצאת בתחילת החלק הניתן לביצוע של התכנית, כך שהקלט של המשתמש יועבר בצורה נוחה לתהליך 2. תרשימים 61 עד 63 מכילים את קוד המקור לפונקציה SelectLibrary.

השתמשנו במספר פונקציות חדשות של OS/2 כדי לתמוך בק/פ עם המשתמש. השתמשנו ב-VioGetCurType וב-VioSetCurType כדי לשלוט בסמן, וב-DosBeep כדי להודיע למשתמש שהוא לחץ על מקש לא נכון.

VioGetCurType ו-VioSetCurType

פונקציות אלו מחזירות וקובעות בהתאמה את הגדרת הסמן הנוכחי. ההגדרה כוללת את המיקום, הגובה, הרוחב ומאפייני התצוגה של הסמן כתו טקסט. לפונקציה זו יש שני פרמטרים.

```
/* Dynamic Link Library Selection Window and Strings**********************************************************/

  char far *DLLWindow[15] = {"┌────────────────────────────────┐",
                             "│ Use  cursor keys to select the │",
                             "│                                │",
                             "│      Dynamic Link Library      │",
                             "│                                │",
                             "│     you would like to use.     │",
                             "│                                │",
                             "│  Press <┘ Enter when complete. │",
                             "├────────────────────────────────┤",
                             "│                                │",
                             "│ * USA Format      (USA.DLL)    │",
                             "│                                │",
                             "│ * European Format (EUROPEAN.DLL)│",
                             "│                                │",
                             "└────────────────────────────────┘"};

  char far *USAStr =      "* USA Format      (USA.DLL)      ";
  char far *EuropeanStr = "* European Format (EUROPEAN.DLL) ";
  int       SelectionFlag = 1;
  int       Selection;

/* VioSetCurType / VioGetCurType Variables *******************************************************************/

  struct CursorData      CursorStructure;
  struct CursorData far *CursorStructurePointer = &CursorStructure;
  unsigned               PreviousAttribute;

/* DosBeep Variables ******************************************************************************************/

  unsigned          Frequency = 750;
  unsigned          Duration  = 100;

/**************************************************************************************************************/
```

תרשים 61. הפונקציה SelectLibrary - הכרזות על משתנים

CursorData הוא מחוון רחוק למבנה המכיל את הפרמטרים המרכיבים את ההגדרה של הסמן המסוים.

VioHandle הוא מלה שמורה המכילה אפסים.

מבנה הנתונים CursorData מכיל ארבעה איברים (שדות) בני מלה אחת. כאשר מצב התצוגה הוא במצב טקסט, ניתן בעזרת שני האיברים הראשונים לשלוט בגודל ובמיקום של הסמן ביחידת התו. איברים אלה נקראים CursorStartLine ו-CursorEndLine, והם מציינים את השורה האופקית בתוך יחידת התו שבה הסמן מתחיל ומסתיים בהתאמה. המלה השלישית נקראת CursorWidth והיא מציינת את רוחב הסמן בטורים. תת-מערכת התצוגה ב-OS/2 תומכת כרגע רק ברוחב של טור אחד. המלה הרביעית במבנה נתונים זה נקראת CursorAttribute והיא מציינת את מאפיין התצוגה של הסמן. בנוסף למאפייני התצוגה המתייחסים למצבי התצוגה השונים, ערך של מינוס אחד (-1) ב-CursorAttribute מצביע על כך, שהסמן צריך להיות סמוי.

DosBeep

פונקציה זו הינה מישק נוח, המאפשר להפיק צליל מהרמקול. לפונקציה שני פרמטרים:

Frequency הוא ערך בגודל מלה שיכול לקבל ערכים מ-37 עד 32,767. הוא מציין את תדירות הצליל ב-Hertz.

```
void SelectLibrary()
{

   ErrorCode = VIOGETCURTYPE(CursorStructurePointer,0);              /* Get the current cursor type  */
   PreviousAttribute = CursorStructure.cur_attribute;                /* and save it so we can restore*/
   CursorStructure.cur_attribute = -1;                               /* it later                     */
   ErrorCode = VIOSETCURTYPE(CursorStructurePointer,0);              /* Then make the cursor hidden  */

   for(RowCounter = 0; RowCounter < DLLWindowLength; RowCounter++)   /* Display the dynamic link     */
                                                                     /* library selection window     */
      ErrorCode = VIOWRTCHARSTRATT(DLLWindow[RowCounter],
                                   VioLength = strlen(DLLWindow[RowCounter]),
                                   Row = 5 + RowCounter,
                                   Column = 22,
                                   Attribute = "\044",
                                   VioHandle);

   ErrorCode = VIOWRTCHARSTRATT(USAStr,                              /* Highlight the current library*/
                                VioLength = strlen(USAStr),          /* selection                    */
                                Row = 15,
                                Column = 24,
                                Attribute = "\137",
                                VioHandle);
```

תרשים 62. הפונקציה SelectLibrary - חלק 1

Duration הוא ערך בגודל מלה המציין את משך הזמן, באלפיות שניה, שבו יש להפיק את הצליל.

פונקציה זו מתבצעת בצורה מסונכרנת ליישום. כתוצאה מכך, השליטה חוזרת לאחר מספר אלפיות השניה, כפי שצוין בפרמטר.

SelectLibrary

תרשים 62 מציג את ההתחלה של הפונקציה SelectLibrary. בתרשים זה העלמנו את הסמן, הצגנו את החלון המנחה, והארנו את הספריה הדינמית הנבחרת כרגע. השתמשנו ב-VioGetCurType כדי לתחל את מבנה הנתונים CursorData עם הגדרת הסמן הנוכחית. לפני שאנו משנים את מאפיין הסמן למינוס אחד (-1), אנו שומרים את הערך הקודם של מאפיין הסמן כדי שנוכל לשחזר את הסמן בסוף השיגרה. לאחר מכן השתמשנו ב-VioSetCurType כדי להעלים את הסמן.

לאחר שהצגנו את חלון בחירת הספריה בעזרת לולאת "for", הצגנו שוב את השורה המצביעה על הספריה USA Format עם מאפייני תצוגה אחרים. כך אנו מאירים בדרך יעילה את השורה הזו, כדי להראות למשתמש את הספריה שכרגע נבחרה.

תרשים 63 הוא השארית של הפונקציה SelectLibrary. הוא מכיל לולאה מסוג "do-while" (לולאת "בצע עד אשר...") בלוגיקה של התכנית בתרשים 60) שמגיבה לקלט של המשתמש מהמקלדת. בתחילה קראנו לפונקציה KbdCharIn המציינת שיש להמתין לתו. הקריאה חוזרת לאחר שנלחץ מקש ואנו בודקים את קוד התו שנלחץ, כדי לראות אם זה היה חץ כלפי מעלה או כלפי מטה. אם נלחץ אחד מהחצים, אנו בודקים איזו ספריה מוארת כרגע, בהתאם לערך המשתנה SelectionFlag. לאחר מכן אנו מעדכנים את ה-SelectionFlag ומאירים את הספריה השויה, כדי להראות שכעת ספריה זו נבחרה. אם המקש שנלחץ לא היה אחד ממקשי החצים, אנו בודקים אם היה זה מקש Enter. אם נלחץ מקש Enter, איננו עושים דבר במשפט ה-"if" מאחר וזה מצב הסיום של

הלולאה הראשית "do-while". משפט "else" האחרון מתבצע אם נלחץ כל מקש אחר, והוא קורא לפונקציית ה-DosBeep כדי להודיע למשתמש על פעולה לא נכונה.

הפונקציה SelectLibrary מסתיימת במחיקת החלון של בחירת הספריה, ובהפעלה מחדש של הסמן. למרות שלא הראנו זאת, ביצענו שינוי בתהליך 1 כדי להפוך את ה-SelectionFlag למחרוזת תווים, והכנסנו אותה כמשתנה המחרוזת הראשון המועבר לתהליך 2 בביצוע הפונקציה DosExecPgm.

```
do {                                                         /* Top of loop that responds to */
   ErrorCode = KBDCHARIN(KeyStructurePointer,                /* user's keyboard input        */
                         IOWait = 0,                         /* IOWait = 0 means block until */
                         KbdHandle = 0);                     /* character received           */

   if ((KeyStructure.scan_code == 0x48)|(KeyStructure.scan_code == 0x50))  /* Check scan code looking for */
                                                             /* up or down cursor keys       */
      if (SelectionFlag == 1) {
         SelectionFlag = 2;
         ErrorCode = VIOWRTCHARSTRATT(USAStr,
                                      VioLength = strlen(USAStr),
                                      Row = 15,
                                      Column = 24,
                                      Attribute = "\044",
                                      VioHandle);
         ErrorCode = VIOWRTCHARSTRATT(EuropeanStr,           /* Depending upon the value of  */
                                      VioLength = strlen(EuropeanStr),  /* SelectionFlag, unhighlight */
                                      Row = 17,              /* one selection and highlight  */
                                      Column = 24,           /* the other                    */
                                      Attribute = "\137",
                                      VioHandle);
      }
      else {
         SelectionFlag = 1;
         ErrorCode = VIOWRTCHARSTRATT(EuropeanStr,
                                      VioLength = strlen(EuropeanStr),
                                      Row = 17,
                                      Column = 24,
                                      Attribute = "\044",
                                      VioHandle);
         ErrorCode = VIOWRTCHARSTRATT(USAStr,
                                      VioLength = strlen(USAStr),
                                      Row = 15,
                                      Column = 24,
                                      Attribute = "\137",
                                      VioHandle);
      }

   else if (KeyStructure.char_code == 0x0D) ;                /* If the Enter key was pressed */
                                                             /* don't do anything yet        */
   else
      ErrorCode = DOSBEEP(Frequency,                         /* Sound a beep if any other key*/
                          Duration);                         /* was pressed                  */

} while (KeyStructure.char_code != 0x0D);                    /* End loop if Enter key was    */
                                                             /* pressed                      */
ErrorCode = VIOSCROLLUP(TopRow=5,
                        LeftCol=22,                          /* Selection is complete, erase */
                        BotRow=20,                           /* the library selection window */
                        RightCol=60,
                        NumLines=-1,
                        (char far *)FillChar,
                        VioHandle = 0);

CursorStructure.cur_attribute = PreviousAttribute;           /* Restore the cursor           */
ErrorCode = VIOSETCURTYPE(CursorStructurePointer,0);

}
/***********************************************************************************************/
```

תרשים 63. הפונקציה SelectLibrary - חלק 2

תהליך 2 החדש

ביצענו מספר שינויים בתהליך 2 והוצאנו את הפונקציות BuildTimeString ו-BuildDateString. הוספנו לוגיקה כדי לחפש את הערך במשתנה המחרוזת, במטרה לקבוע באיזו ספריה דינמית להשתמש. ולבסוף, הוספנו הכרזות ופונקציות ההכרחיות להפעלת קישור דינמי בזמן ריצה.

בחלק העליון של תרשים 64, תבחין בוודאי שהכרזנו שוב על פונקציות סטנדרטיות בשפת C לזמן ריצה (itoa, atoi וכו'), כדי להשתמש במחוונים רחוקים לפרמטרים של המחוון. אנו עושים זאת, מכיון שאנו בונים את התכנית תוך שימוש במודל הרחב בשפת C.

```
/*****************************************************************************************/
/*  Multiple Processes with Dyananic Linking - Process 2                                  */
/*****************************************************************************************/

#include <doscall.h>                                  /* OS/2 API dynanic link library*/
#include <stdio.h>                                    /* C standard I/O run tine lib  */
#include <string.h>                                   /* C string library             */

  char * cdecl itoa(int, char far *, int);            /* These standard library       */
  int cdecl atoi(const char far *);                   /* declarations have been       */
  size_t cdecl strlen(const char far *);              /* modified to take far pointer */
  char * cdecl strcat(char far *, const char far *);  /* paraneters for use with large*/
  int cdecl printf(const char far *, ...);            /* model programning            */

/*****************************************************************************************/
/* These external reference declarations are used with load tine dynanic linking.        */
/* They have been connented out of the source code but are included as a reference.      */

/*
extern void far pascal BuildTimeString (
  unsigned char,
  unsigned char,
  unsigned char,
  char far *);

extern void far pascal BuildDateString (
  unsigned char,
  unsigned char,
  unsigned,
  char far *);
*/

/* Display window character definitions ***************************************************/

  char far *Proc2Window[12] = {"┌──────────────────────┐",   /* An array of strings used to */
                               "│                      │",   /* outline the Process 1 display*/
                               "├──────────────────────┤",   /* window                      */
                               "│                      │",
                               "│                      │",
                               "│                      │",
                               "│                      │",
                               "│                      │",
                               "│                      │",
                               "│                      │",
                               "│                      │",
                               "└──────────────────────┘"};

  int          RowCounter = 0;                        /* Variables used in "for" loop */
  int          WindowLength = 12;

/* General Variables - used throughout the progran ****************************************/

  unsigned     ErrorCode;                             /* Error code return from OS/2  */
                                                      /* function calls               */
/* VioWrtCharStrAtt Variables - used to display text prompts and messages ****************/

  char far     *CharStr;                              /* String to be written         */
  unsigned     VioLength;                             /* Length of character string   */
  unsigned     Row;                                   /* Starting position - row      */
  unsigned     Column;                                /* Starting position - column   */
  char far     *Attribute;                            /* Display attribute            */
  unsigned     VioHandle = 0;                         /* Reserved word of zeros       */
```

תרשים 64. תהליך 2 - הכרזות משתנים (חלק 1)

לאחר מכן תראה שתי הגדרות של הפניות חיצוניות לפונקציות BuildTimeString ו-BuildDateString. שים לב שהפכנו את ההגדרות החיצוניות האלו להערות. במהלך הניפוי הראשוני השתמשנו בפונקציות אלו בשיטת הקישור הדינמי בעת הטעינה. אף על פי שהן לא מופעלות בשיטת קישור דינמי בעת הריצה, החלטנו להשאיר אותן, כדי להראות איך ניתן לעשות זאת. ספרנית היבוא (תכנית השירות IMPLIB) יכולה לעבד קובץ להגדרת המודול הדומה לזה הנמצא בתרשים 72 כדי ליצור ספריית יבוא. ספריית יבוא זו, יחד עם ההכרזות החיצוניות המוצגות בתרשים 64, מאפשרות להשתמש בקישור דינמי בעת טעינה, כדי לגשת לפונקציות בספריות הדינמיות המשמשות אותנו בדוגמה של קישור דינמי בעת ריצה.

```
/* VioScrollUp Variables ******************************************************************/

   unsigned           TopRow;                                   /* Upper left hand corner    */
   unsigned           LeftCol;
   unsigned           BotRow;                                   /* Bottom right hand corner  */
   unsigned           RightCol;
   unsigned           NumLines;                                 /* Number of lines to scroll */
   char               FillChar[2] = {0x20,0x1F};                /* Fill character to use     */

/* DosRead Variables **********************************************************************/

   unsigned           ReadHandle;                               /* File handle(pipe read handle)*/
   char               CharacterCell[2];                         /* Input buffer              */
   unsigned           BufferLength = 2;                         /* Input buffer length       */
   unsigned           BytesRead;                                /* Bytes read - returned     */

/* DosGetDateTime Variables ***************************************************************/

   struct DateTime    CurDateTime;                              /* Date and Time structure   */
   struct DateTime far *CurDateTimePointer = &CurDateTime;      /* Date/Time structure pointer */

/* DosSleep Variables *********************************************************************/

   unsigned long      TimeInterval;                             /* Sleep time duration       */

/* DosLoadModule Variables ****************************************************************/

   char               ObjNameBuf[32];                           /* Object name buffer        */
   unsigned           ObjNameBufLength;                         /* Object name buffer length */
   char far          *ModuleName;                               /* DynaLink module name      */
   unsigned           ModuleHandle;                             /* Module handle - returned  */

/* DosGetProcAddr Variables ***************************************************************/

   char far          *ProcName;                                 /* DynaLink procedure name   */
   void (far pascal  *ProcAddress)();                           /* Procedure address - returned */

   void (far pascal  *BuildTimeString)();
   void (far pascal  *BuildDateString)();

/* String variables ***********************************************************************/

   char               Time[9] = "        \0";                   /* Used to build the time string*/
   char              *TimePointer = Time;

   char               Date[11] = "          \0";                /* Used to build the date string*/
   char              *DatePointer = Date;

   char               LogString[22];                            /* Used to build the log string */
   char              *LogStringPointer = LogString;

/* DosExit Variables **********************************************************************/

   unsigned           ActionCode=1;                             /* Exit all threads in process */
   unsigned           ResultCode=0;                             /* Result saved for DosCWait */

/******************************************************************************************/

   void far           DateTimeProcedure();                      /* Function declaration      */

/******************************************************************************************/
```

תרשים 65. תהליך 2 - הכרזות משתנים (חלק 2)

בתרשימים הבאים אפשר לראות שינויים בהכרזות של משתנים בתהליך 2. שינויים אלה כוללים השמטה של ההכרזות המשמשות אך ורק את הפונקציות BuildTimeString ו-BuildDateString, אשר הועברו למודולים המקושרים בצורה דינמית. תרשימים 64 עד 68 מכילים את קוד המקור של תהליך 2, הכתוב בשפת C.

לפני שנעבור על הגירסה המשופרת של תהליך 2, נסקור תחילה את פונקציות OS/2 החדשות המשמשות בדוגמה זו.

```
main(argc, argv, envp)                                              /* Start of C main routine      */
int argc;
char far *argv[ ];
char far *envp[ ];

{

/* DosCreateThread Variables ***********************************************************************/

  unsigned          ThreadIDWord;                                   /* New thread ID                */
  unsigned char     NewThreadStack[100];                            /* New thread stack             */

/* Start of executable program *********************************************************************/

  ErrorCode = VIOWRTCHARSTRATT(CharStr = "PROCESS 2",               /* Label area of screen used    */
                         VioLength = strlen(CharStr),               /* for Process 2                */
                         Row = 5,
                         Column = 55,
                         Attribute = "\032",
                         VioHandle = 0);

  for(RowCounter =0; RowCounter < WindowLength; RowCounter++)       /* Display the Process 2 window */

    ErrorCode = VIOWRTCHARSTRATT(Proc2Window[RowCounter],
                           VioLength = strlen(Proc2Window[RowCounter]),
                           Row = 7 + RowCounter,
                           Column = 48,
                           Attribute = "\032",
                           VioHandle);

  if (atoi(argv[0]) == 1) {                                         /* Library selection passed in  */
                                                                    /* argv[0]                      */
    ErrorCode = DOSLOADMODULE((char far *)ObjNameBuf,
                         ObjNameBufLength,                          /* USA format uses library      */
                         ModuleName = "USA",                        /* module the filename USA.DLL  */
                         (unsigned far *)&ModuleHandle);

    ErrorCode = VIOWRTCHARSTRATT(CharStr = "USA Format",
                           VioLength = strlen(CharStr),
                           Row = 20,
                           Column = 54,
                           Attribute = "\032",
                           VioHandle);
  }

  else {

    ErrorCode = DOSLOADMODULE((char far *)ObjNameBuf,
                         ObjNameBufLength,
                         ModuleName = "EUROPE",                     /* European format uses library */
                         (unsigned far *)&ModuleHandle);            /* module with the filename     */
                                                                    /* EUROPE.DLL                   */
    ErrorCode = VIOWRTCHARSTRATT(CharStr = "European Format",
                           VioLength = strlen(CharStr),
                           Row = 20,
                           Column = 52,
                           Attribute = "\032",
                           VioHandle);
  }
```

תרשים 66. תהליך 2 - פונקציה ראשית (חלק 1)

פונקציה זו טוענת מודול המקושר דינמית לזיכרון ומחזירה מקשר למודול. המקשר משמש כהפניה למודול בכל אחת מפונקציות הקישור הדינמי שבאות לאחר מכן. יש לפונקציה זו ארבעה פרמטרים.

ObjNameBuf הוא מחוון רחוק למאגר תווים. אם הקריאה לפונקציה תכשל, מערכת ההפעלה תציב בו את השם של האובייקט שגרם לתקלה.

ObjNamBufL הוא מלה המכילה את אורך המאגר המקבל את שם האובייקט.

ModuleName הוא מחוון רחוק למחרוזת תווים המכילה את השם של המודול הדינמי. מחרוזת זו יכולה להיות באורך שיכול לנוע מאחד עד שמונה תווים. ההרחבה של שם הקובץ היא DLL. והקובץ חייב להיות מאוחסן באחד המדריכים הנמצא במסלול החיפוש של הספריה.

```
ErrorCode = DOSGETPROCADDR(ModuleHandle,                        /* The dynamic link modules are */
                           ProcName = "BUILDTIMESTRING",        /* loaded and we must get the   */
                           (unsigned long far *)&BuildTimeString); /* two procedure addresses that */
                                                                /* are used to call the dynalink*/
ErrorCode = DOSGETPROCADDR(ModuleHandle,                        /* procedures                   */
                           ProcName = "BUILDDATESTRING",
                           (unsigned long far *)&BuildDateString);

ErrorCode = DOSCREATETHREAD(DateTimeProcedure,                  /* Create Thread 2 for date and */
                            (unsigned far *)&ThreadIDWord,      /* time procedure               */
                            (unsigned char far *)&NewThreadStack[98]);

ReadHandle = atoi(argv[1]);                                     /* Get the pipe read handle from*/
                                                                /* the second argument string   */
do{
                                                                /* Top of loop that continues   */
   ErrorCode = DOSREAD(ReadHandle,                              /* reading character cells from */
                       (char far *)CharacterCell,               /* the pipe until a "q"         */
                       BufferLength,                            /* character is received        */
                       (unsigned far *)&BytesRead);

   ErrorCode = VIOSCROLLUP(TopRow=10,                           /* Scroll the window with the   */
                           LeftCol=49,                          /* new attribute                */
                           BotRow=17,
                           RightCol=69,
                           NumLines=1,
                           (char far *)CharacterCell,
                           VioHandle = 0);

   LogString[0] = CharacterCell[0];                             /* Build the log string         */
   LogString[1] = 0x00;
   LogStringPointer = strcat(LogStringPointer," ");
   LogStringPointer = strcat(LogStringPointer,TimePointer);
   LogStringPointer = strcat(LogStringPointer," ");
   LogStringPointer = strcat(LogStringPointer,DatePointer);

   ErrorCode = VIOWRTCHARSTRATT((char far *)LogStringPointer,   /* Display the new log string   */
                                VioLength = strlen(LogStringPointer),
                                Row = 17,
                                Column = 49,
                                Attribute = &CharacterCell[1],
                                VioHandle);

}
while(CharacterCell[0] != 0x71);                                /* Test for a "q" character     */

ErrorCode = DOSFREEMODULE(ModuleHandle);                        /* Free module from memory      */

DOSEXIT(ActionCode,                                             /* Notify OS/2 of termination   */
        ResultCode);

}
/************************************************************************************************/
```

תרשים 67. תהליך 2 - פונקציה ראשית (חלק 2)

```
void far DateTimeProcedure()
{

  for(;;){                                                /* Begin endless loop            */

     ErrorCode = DOSGETDATETIME(CurDateTimePointer);      /* Get the date and time         */

     (*BuildTimeString)(CurDateTime.hour,                 /* Call BuildTimeString          */
                   CurDateTime.minutes,
                   CurDateTime.seconds,
                   (char far *)TimePointer);

     ErrorCode = VIOWRTCHARSTRATT(TimePointer,            /* Display the time string       */
                          VioLength = strlen(TimePointer),
                          Row = 8,
                          Column = 51,
                          Attribute = "\032",
                          VioHandle);

     (*BuildDateString)(CurDateTime.month,                /* Call BuildDateString          */
                   CurDateTime.day,
                   CurDateTime.year,
                   (char far *)DatePointer);

     ErrorCode = VIOWRTCHARSTRATT(DatePointer,            /* Display the date string       */
                          VioLength = strlen(DatePointer),
                          Row = 8,
                          Column = 60,
                          Attribute = "\032",
                          VioHandle);

     ErrorCode = DOSSLEEP(TimeInterval);                  /* Sleep for about a second      */

  }                                                       /* This thread terminates when  */
                                                          /* the process terminates        */
}
/***************************************************************************************************/
```

תרשים 68. תהליך 2 - DateTimeProcedure

ModuleHandle הוא מחוון רחוק למלה שבה מערכת ההפעלה מציבה את המקשר למודול.

המקשר שחוזר משמש לציון המודול בכל אחת מהקריאות הבאות לפונקציות DosGetProcAddr, DosGetModName ו-DosFreeModule.

DosGetProcAddr

פונקציה זו מחזירה כתובת רחוקה לפרוצדורה מסוימת במודול הדינמי. יש לפונקציה שלושה פרמטרים.

ModuleHandle הוא מלה המכילה את המקשר שחזר מהפונקציה DosLoadModule. הוא מזהה את המודול המסוים שנטען קודם לכן מהזיכרון.

ProcName הינו מחוון רחוק למחרוזת תווים המכילה את השם של פרוצדורה בתוך המודול הדינמי. אם ערך הבורר במחוון זה מכיל אפסים (כלומר ריק), החלק של הכתובת יחסית במחוון מוגדר כך שהוא יקבל את המספר הסידורי של הפרוצדורה הרצויה. המספר הסידורי הוא מספר נקודת הכניסה בתוך המודול.

ProcAddress הינו מחוון רחוק לכתובת רחוקה (מלה כפולה) שלתוכה מציבה מערכת ההפעלה את הכתובת של הפרוצדורה המבוקשת. כתובת זו משמשת לקריאה לפרוצדורה.

קישור דינמי המשתמש במספרים סידוריים במקום בשמות הפרוצדורות הוא קצת יותר יעיל, והוא גם המנגנון היחיד שמאפשר כניסה לפונקציות הנתמכות על-ידי המודול DOSCALLS של OS/2. המספר הסידורי הינו מספר שלם המציין את מיקום הפרוצדורה בתוך רצף של פרוצדורות הנמצאות במודול. משתמשים במספרים הסידוריים המתאימים של פונקציות OS/2 כאשר מודול היעד של היישום מקושר עם DOSCALLS.LIB.

DosFreeModule

הפונקציה מודיעה למערכת ההפעלה שתהליך זה גמר להשתמש במודול הדינמי שצוין. לפונקציה זו יש פרמטר אחד.

ModuleHandle הוא מלה המכילה את המקשר המוחזר עם DosLoadModule, אשר מזהה את המודול הדינמי המסוים שהוטען קודם לזיכרון.

המקשר למודול מפסיק להתקיים לאחר שהקריאה לפונקציה זו הושלמה. אם אין לתהליכים אחרים מקשר למודול זה, שהושג דרך DosLoadModule או DosGetModHandle, אז המודול נמחק מזיכרון המערכת.

תיאור של תהליך 2

תהליך 2 מתחיל במתן שם לשטח שלו במסך והעתקה של מערך המחרוזת לתצוגה שמגדירה את החלון שלו, כפי שראינו בדוגמה העוסקת במספר תהליכים.

הקטע הראשון של הפקודות החדשות קובע איזו ספריה דינמית יש להטעין. כפי שהזכרנו קודם, דוגמה זו נכתבה על מנת לתמוך בשתי ספריות שונות בלבד. שים לב שהשתמשנו במחרוזת המשתנה הראשונה, כדי להעביר את בחירת המשתמש לגבי הספריה. באופן נורמלי משתנה זה, argv[0], מכיל את השם המשמש להתחלת תהליך-הבן כפי שהוצג בדוגמה הקודמת. לאחר מכן שורשרו מספר משתנים למחרוזת ASCIIZ יחידה, ותהליך-הבן ניתח את המחרוזת כדי לחלוץ את המשתנים שהוא צריך. אין אנו משתמשים במחרוזת המכילה את שם התכנית בתהליך 2. כדי לפשט, אנו מעבירים סימן במחרוזת המשתנים הראשונה, שמצביע על הספריה שאותה יש להטעין, במקום להעביר מחרוזת עם שם התכנית.

מחרוזת עם התו "1" ב-argv[0] מצביעה על כך שיש להשתמש בספריה "USA Format". אם argv[0] אינו מכיל את המחרוזת "1", אנו מניחים שהוא מכיל את המחרוזת "2" המצביעה על כך שיש להשתמש בספריה "European Format". בהתאם לתוצאת הבדיקה של argv[0], נטענת הספריה הדינמית המתאימה בעזרת DosLoadModule. לאחר שנבחרה הספריה אנו מציגים מחרוזת המודיעה בצורה ברורה באיזה פורמט נשתמש.

המקשר החוזר מהקריאה לפונקציה DosLoadModule משמש בשתי הקריאות הבאות ל-DosGetProcAddr. פונקציות אלו משיגות את הכתובות של נקודות הכניסה לפונקציות שמשתמשים בהן בדוגמה זו.

יתרת הפקודות בתהליך 2 החדש זהות לאלו הנמצאות בדוגמה שבפרק 4. ההבדלים היחידים הם התחביר המשמש לקריאה לפונקציות BuildTimeString ו-BuildDateString ול-DosFreeModule, כדי לשחרר את המודול הדינמי כאשר התהליך מסתיים.

השתמשנו בפקודה CL הבאה כדי לבצע הידור של התכנית.

cl /c /G2 /Alfu /Fs proc2blb.c

האופציות בשורת הפקודה מגדירות את הדברים הבאים:

/c מבצע את שלב ההידור בדרך כזו, שלאחריו נצטרך לקרוא במפורש לתכנית הקישור.
/G2 מאפשר הפעלת מערכת ההוראות של ה-80286.
/Alfu מאפשר שימוש במודל הרחב של שפת C.
/Fs יוצר תדפיס של תכנית המקור.

השתמשנו בפקודה Link הבאה כדי לקשור את תהליך 2.

link proc2,proc2,,llibc5.lib llibc.lib doscalls.lib;

נתקדם בלימוד ונסתכל במה שעשינו, כדי להפוך את שתי הפונקציות למודול ספריה דינמי נפרד.

ספריית קישור דינמי

ישנם מספר הבדלים משמעותיים בין הפקודות המופיעות בתרשימים 69 עד 71, לבין הפקודות ששימשו אותנו בביצוע של אותם פונקציות בדוגמה הקודמת.

```
/*******************************************************************************************/
/* USA.DLL - USA format dynamic link library                                               */
/*******************************************************************************************/

#include <doscall.h>                                      /* OS/2 API dynamic link library*/
#include <stdio.h>                                        /* C standard I/O run time lib  */
#include <string.h>                                       /* C string library             */

  char * cdecl itoa(int, char far *, int);                /* These standard library       */
  int cdecl atoi(const char far *);                       /* declarations have been       */
  size_t cdecl strlen(const char far *);                  /* modified to take far pointer */
  char * cdecl strcat(char far *, const char far *);      /* parameters for use with large*/
                                                          /* model programming            */

  int _acrtused = 0;                                      /* Allows for no main function  */

  char            TimeHour[3];                            /* String variables used to     */
  char           *TimeHourPointer = TimeHour;             /* build the time string        */
  char            TimeMinutes[3];
  char           *TimeMinutesPointer = TimeMinutes;
  char            TimeSeconds[3];
  char           *TimeSecondsPointer = TimeSeconds;

  char            DateDay[3];                             /* String variables used to     */
  char           *DateDayPointer = DateDay;               /* build the date string        */
  char            DateMonth[3];
  char           *DateMonthPointer = DateMonth;
  char            DateYear[5];
  char           *DateYearPointer = DateYear;

  int             Radix;

/*******************************************************************************************/
```

תרשים 69. USA.DLL - הכרזות

```
extern void far pascal BuildTimeString(Hour, Minutes, Seconds, TimePointer)     /* This function formats the    */
unsigned char Hour;                                                              /* time suitable for display    */
unsigned char Minutes;
unsigned char Seconds;
char far *TimePointer;
{

  Radix = 10;

  TimeHourPointer = itoa(Hour,TimeHour,Radix);                                   /* Format the hour              */
  if(strlen(TimeHourPointer) == 1){
     TimePointer[0] = 0x30;
     TimePointer[1] = TimeHourPointer[0];
  }
  else{
     TimePointer[0] = TimeHourPointer[0];
     TimePointer[1] = TimeHourPointer[1];
  }

  TimePointer[2] = 0x3A;                                                         /* Format the minutes           */
  TimeMinutesPointer = itoa(Minutes,TimeMinutes,Radix);
  if(strlen(TimeMinutesPointer) == 1){
     TimePointer[3] = 0x30;
     TimePointer[4] = TimeMinutesPointer[0];
  }
  else{
     TimePointer[3] = TimeMinutesPointer[0];
     TimePointer[4] = TimeMinutesPointer[1];
  }

  TimePointer[5] = 0x3A;                                                         /* Format the seconds           */
  TimeSecondsPointer = itoa(Seconds,TimeSeconds,Radix);
  if(strlen(TimeSecondsPointer) == 1){
     TimePointer[6] = 0x30;
     TimePointer[7] = TimeSecondsPointer[0];
  }
  else{
     TimePointer[6] = TimeSecondsPointer[0];
     TimePointer[7] = TimeSecondsPointer[1];
  }

}
/*************************************************************************************************************************/
```

תרשים 70. USA.DLL - הפונקציה BuildTimeString

שמת לב בוודאי לכך, שהפונקציה הראשית של שפת C חסרה בספריה הדינמית. הפונקציה הראשית מגדירה את נקודת הכניסה הראשונה לתכנית כאשר התכנית נטענת לזיכרון. הטכניקה שאיפשרה לנו לבצע הידור וקישור מבלי להשתמש בפונקציה הראשית, היא על-ידי הגדרה של משתנה actrused_ שהוא מספר שלם, והצבה של הערך אחד. משתנה זה הוא חלק מהתמיכה של C/2 בזמן ריצה. הפרטים עליו הם מעבר להיקף של ספר זה.

תבחין בוודאי בכך שבתרשימים 69 עד 71 הכרזנו שוב על פרמטרים של שגרות הספריה הסטנדרטיות שמשתמשים בהם בתכנית זו, הבנויה לפי המודל הרחב.

פקודת CL שהשתמשנו בה כדי להדר את המודול הדינמי היא:

cl /c /G2 /Gs /Alfu /Fs usa.c

כאשר Gs/ משמש להוצאת ה-stack probes.

גם הפקודה Link שונה במידה מעטה:

link usa.obj,usa.dll,,llibc5.lib llibc.lib doscalls.lib,usa.def

```
extern void far pascal BuildDateString(Month, Day, Year, DatePointer)        /* This function formats the     */
unsigned char Month;                                                          /* date suitable for display     */
unsigned char Day;
unsigned Year;
char far *DatePointer;
{

  Radix = 10;

  DateMonthPointer = itoa(Month,DateMonth,Radix);                            /* Format the month              */
  if(strlen(DateMonthPointer) == 1){
     DatePointer[0] = 0x30;
     DatePointer[1] = DateMonthPointer[0];
  }
  else{
     DatePointer[0] = DateMonthPointer[0];
     DatePointer[1] = DateMonthPointer[1];
  }

  DatePointer[2] = 0x2F;                                                     /* Format the day                */
  DateDayPointer = itoa(Day,DateDay,Radix);
  if(strlen(DateDayPointer) == 1){
     DatePointer[3] = 0x30;
     DatePointer[4] = DateDayPointer[0];
  }
  else{
     DatePointer[3] = DateDayPointer[0];
     DatePointer[4] = DateDayPointer[1];
  }

  DatePointer[5] = 0x2F;                                                     /* Format the year               */
  DateYearPointer = itoa(Year,DateYear,Radix);
  DatePointer[6] = DateYearPointer[2];
  DatePointer[7] = DateYearPointer[3];

}
/*************************************************************************************************************/
```

תרשים 71. USA.DLL - הפונקציה BuidDateString

ההבדל בפקודת קישור זו הוא בכך, שנתנו שם למודול המתקבל עם הסיומת DLL. אנו גם ציינו קובץ להגדרת המודול. ניתן לראות קובץ זה בתרשים 72.

הקובץ להגדרת המודול מציין שזו שיגרת ספריה, אשר מופעלת במצב מוגן. הוא כולל תיאור ומגדיר את שתי המחרוזות עם שמות הפרוצדורות שיש לייצא. השמות הם: BUILDTIMESTRING ו-BUILDDATEESTRING.

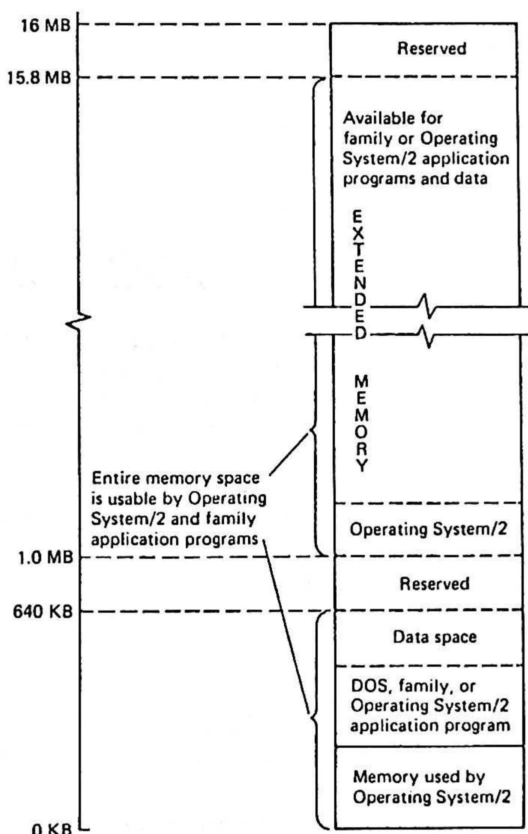
```
;
; Module Definition File
; MYLIBRARY Dynamic Link Library
; Created by JIK, AMM, and RLW
; 1987
;
LIBRARY
PROTMODE
DESCRIPTION 'MYLIBRARY Ver 1.0 Copyright 1987 JIK,AMM,RLW'
EXPORTS
   BUILDTIMESTRING
   BUILDDATESTRING
```

תרשים 72. קובץ להגדרת המודול

כפי שהסברנו קודם, אם אנו רוצים להשתמש בספריה הדינמית עם קישור דינמי בזמן טעינה, אנו צריכים להשתמש באותו קובץ להגדרת המודול עם ספרנית היבוא כדי לייצור ספריית יבוא. משפט ה-implib ייראה במקרה זה כך:

implib usa.lib usa.def

אנו ניתן לספריה שם, למשל "datetime.lib", או כל קיצור אחר. ספריית יבוא זו יכולה להיות מקושרת בצורה דינמית לכל קבוצה של מחרוזות תאריך וזמן היוצרות מודולים המקושרים בצורה דינמית, כל עוד נשמר נוהל ההעברה של הפרמטרים.



ניהול הזיכרון ע"י OS/2 במצב מוגן
אפשר לנצל עבור יישומים את כל הטווח של 16MB פרט לשטח השמור

פרק 8

תמיכה בסביבה בינלאומית

בפרק 1 הצגנו את התמיכה בשפות לאומיות ואת הכלים לטיפול בהודעות של OS/2. מערכת ההפעלה הוצאה לאור באחת עשרה שפות, עם תמיכה התחלתית בחמישה דפי קוד ומדפסות. היא גם תומכת במידע אחר הקשור ל-17 מדינות. ברור מכל האמור לעיל, שמערכת הפעלה כזו מיועדת לטפל בצרכים של ציבור בינלאומי. האמצעים ב-OS/2 שמספקים את הפונקציות הללו הם שירותי התמיכה בשפות לאומיות (National Language Support - NLS). כדי לתמוך בדרישות של ציבור זה, OS/2 מספקת שירותי מערכת לטיפול בהודעות, אשר מסייעות לתרגום הודעות התוכנה לשפות שונות. בפרק זה אנו מביאים סקירה קצרה על האמצעים השונים האלה התומכים בסביבה בינלאומית. נתחיל בסקירה על התכנית לתרגום שפות ולאחר מכן נעבור ל-API המטפלים בהודעות ולפונקציות ה-NLS.

התמיכה של OS/2 בשפות לאומיות

OS/2 יצאה לאור באחת עשרה שפות שונות. מאחר ו-OS/2 פותחה באנגלית, זוהי הגירסה הראשונה שיצאה לשוק. בעשר הגירסאות האחרות המיועדות לשאר השפות, כל ההודעות המוצגות על-ידי מערכת ההפעלה, כמו גם הפרסומים שקשורים למערכת (מדריך למשתמש, ספר טכני וכו') תורגמו לאותן שפות. רשימת השפות ניתנה בפרק 1.

כדי לעזור בתהליך התרגום, כל התמליל של ההודעות נשמר בנפרד ממרכיבי מערכת ההפעלה. מערכת של אמצעים לטיפול בהודעות תומכת בגישה של מערכת ההפעלה להודעות אלו. אמצעים אלה נכתבו בצורה כללית, והמנגנונים והמישקים שלהם פורסמו כדי להקל על מפתחי תוכנה עצמאיים להוסיף יישומים ותת-מערכות בשפות שונות.

אמצעים לטיפול בהודעות

האמצעים לטיפול בהודעות מורכבים מקבוצה של תכניות שירות וקבוצה של פונקציות. ניתן להשתמש בחלק מתכניות השירות האלה רק בזמן תהליך הפיתוח של היישום והן מסופקות רק עם ערכת הכלים למתכנת ב-OS/2. שאר תכניות השירות וכל מישקי הפונקציות הם מאפיינים סטנדרטיים של מערכת ההפעלה הבסיסית.

האמצעים לטיפול בהודעות כוללים שלוש תכניות שירות. **MKMSGF** הינה תכנית שירות להכנת הודעות, ההופכת קובץ של הודעות לצורה שבה ניתן להשתמש בו עם פונקציות ותכניות שירות אחרות לטיפול בהודעות. תכנית שירות זו היא

חלק מערכת הכלים למתכנת (Toolkit). MSGBIND הינה תכנית שירות, אשר מוסיפה הודעות למודול של תכנית הניתן לביצוע (קובץ EXE.). תכנית שירות זו גורמת להודעות להמצא דרך קבע בזיכרון. לכן, מערכת ההפעלה אינה צריכה לעבור דרך מערכת הקבצים כדי לשלוף הודעה מהקובץ. גם תכנית שירות זו מסופקת כחלק של ערכת הכלים למתכנת. HELPMSG היא תכנית להודעות עזר. המשתמש מבצע אותה כדי לקבל תיאור מפורט יותר של ההודעה הקצרה המוצגת על-ידי המערכת או היישום. בכך, היא מאפשרת למפתח היישומים לספק שתי רמות של הודעות. תכנית שירות זו מגיעה כחלק סטנדרטי של מערכת ההפעלה.

יש שלוש פונקציות דינמיות התומכות בהודעות. DosGetMessage היא פונקציה דינמית שמאפשרת ליישום לשלוף הודעה. לגבי היישום מקור ההודעה שקוף לחלוטין, והוא יכול להיות הזיכרון, או קובץ בדיסק. הפונקציה גם מציבה משתנים בהודעות, אם קיימים בה כאלה. DosInsMessage מכניסה משתנה לתוך הודעה שנשלפה קודם. DosPutMessage מוציאה הודעה לקובץ או להתקן. הפלט כולל מאפיין של גלילת שורות, אשר עורך את ההודעה כתצוגה ב-80 טורים.

ועכשיו, הבה נסתכל בכל אחת מהפונקציות האלו, ונתחיל בתכניות השירות.

MKMSGF הופכת קובץ של הודעות לצורה המאפשרת שימוש עם פונקציות ותכניות שירות אחרות לטיפול בהודעות. תכנית שירות זו היא חלק מערכת הכלים למתכנת. היא מקבלת כקלט שני פרמטרים: שם קובץ הקלט ושם קובץ הפלט. קובץ הקלט מכיל הודעות בתבנית מיוחדת שניתנת לתרגום על-ידי תכנית שירות זו. קובץ הפלט מכיל את ההודעות בצורה שבה ניתן להשתמש בהן עם פונקציות ותכניות שירות אחרות לטיפול בהודעות.

לקובץ ההודעות יש שלושה חלקים:

1. מספר משתנה של שורות הערה, שניתן להשתמש בהן כדי לזהות את הקובץ ולתעד אותו.
2. מזהה בן שלושה תווים, שמזהה את היישום (או תת-מערכת) שמשתמש בקובץ הודעות זה.
3. ההודעות עצמן.

```
;
; This is the message file for the XYZ program
; Created by JIK, AMM, and RLW
;
XYZ
XYZ0000W: Warning! Fixed disk %1 is about to be erased!
XYZ0001P: Please enter your name: %0
XYZ0002E: Can't find file %1
XYZ0003E:
XYZ0004E:
XYZ0005H: Name format is Lastname, First MI
XYZ0006H:
XYZ0007I: %1 records created
```

תרשים 73. דוגמה של קובץ הודעות

בתרשים 73 ניתן לראות דוגמה של קובץ ההודעות. כל הודעה מורכבת מכותרת שבעקבותיה בא התוכן. כותרת ההודעה כוללת את קוד הזיהוי של היישום, מספר הודעה בן 4 ספרות ותו נוסף המשמש לסיווג ההודעה. קובץ הודעות יחיד יכול להכיל כ-6,000 הודעות. ההודעות מסווגות לפי:

- שגיאה - E (Error).
- עזרה - H (Help).
- מידע - I (Information).
- הנחייה - P (Prompt).
- אזהרה - W (Warning).

יש לסיים את הכותרת בנקודתיים וברווח (:).

לאחר הכותרת בא תוכן ההודעה, שיכול לכלול עד תשעה משתנים (המשתנה נקרא גם "מזהה לתוכן משתנה") לצורך תוכן משתנה של ההודעות. היכולת להשתמש בהודעות עם תוכן משתנה מאפשר לתכניות להחזיק מספר מינימלי של הודעות. התוכן של ההודעה מוכנס תוך כדי ביצוע. תשעת משתני ההודעות מאופיינים על-ידי צירופי התווים 9% ,... ,2% ,1%.

צירוף התווים 0% משמש כסוף שורת ההודעה, כדי לבטל את ההשפעה של carriage return ו-line feed. התווים 0% מאפשרים למשתמש להכניס נתונים, בהמשך להודעה שקיבל. קובץ ההודעות שבתרשים 73 מכיל הודעות עם משתנים שניתן להחליף אותם בכל מידע אחר תוך כדי ביצוע.

שים לב לתרשים 73 וראה שמספרי ההודעות חייבים להיות רציפים ושלמים. למרות שהספרות הריקות חייבות להכיל אפסים, אין המספר הראשון חייב להיות 0000.

שם קובץ ההודעות חייב למלא אחר כללים מסוימים. הוא חייב להיות בן שלושה תווים ולהיות שווה לצירוף התווים המזהה את היישום. הסיומת חייבת להיות הסיומת הסטנדרטית MSG. מפתחי יישומים צריכים לנסות ולהגדיר את הזיהוי של היישום בצורה כזו, שיהיה ייחודי וקל לזכירה. כפילות בשמות של קבצי הודעות אינה מהווה בעיה, כל עוד היישומים נמצאים במדריכים נפרדים והם משתמשים בשם הקובץ (של ההודעות) המלא, כולל מסלול, כדי לשלוף הודעה מהקובץ. לדוגמה, בתרשים 73 המזהה הוא XYZ, ולכן שם קובץ ההודעות יהיה XYZ.MSG.

MSGBIND מוסיפה הודעות למודול הטעינה (קובץ EXE). היא מסופקת כחלק של מערכת הכלים למתכנת. ישנם שלושה קבצים שמעורבים בתהליך הקשירה: קובץ הודעות (MSG.), קובץ הניתן לביצוע (EXE.) וקובץ ASCII מיוחד שמכיל את את ההנחיות לתכנית השירות לקשירת הודעות. תכנית שירות זו קוראת קובץ ASCII כדי לקבוע אילו הודעות מתוך קובץ ההודעות יש לקשור לאיזה מודול טעינה. בתרשים 74 תמצא דוגמה של קובץ ASCII המשמש כקלט לתכנית השירות.

בתרשים 74, השתמשנו בסימן "גדול מ-" (<) כדי להורות לתכנית השירות ששורה זו מכילה את השם של מודול הטעינה, שאליו יש לחבר את הודעות העוקבות. הסימן "פחות מ-" (>) מצביע על השורה שמכילה את שם קובץ ההודעות שממנו יש לשלוף את ההודעות העוקבות. ההודעות שצוינו מועתקות לאחר מכן לתוך סגמנט הודעות, אשר מסופח לקובץ ה-EXE.

```
>c:\codepath\XYZ123A.EXE
<C:\message\XYZ.MSG
XYZ0000
XYZ0002
>c:\codepath\XYZ123B.EXE
XYZ0000
XYZ0002
```

תרשים 74. דוגמה של קובץ קלט לתכנית לקשירת הודעות

בהפעלה אחת של תכנית שירות זו ניתן לקשור הודעות מכמה קבצי הודעות, וניתן לחבר אותן לכמה מודולי טעינה. התהליך הזה מתקיים באמצעות קובץ ASCII המשמש כקובץ קלט לתכנית השירות, ובו מופיעים שמות קבצים, שמות קבצי ההודעות ומספרי הודעות חדשים. תכנית השירות תחפש הודעות בקובץ ההודעות האחרון שצוין, ותוסיף אותן לסגמנט ההודעה של מודול הטעינה האחרון שצוין.

HELPMSG מופעלת על-ידי המשתמש כדי לגשת להודעה מורחבת ולקבל תיאור מפורט יותר של הודעה קצרה שהוצגה על-ידי המערכת, או על-ידי היישום. פונקציה זו מגיעה כחלק מתכניות השירות הסטנדרטיות של OS/2, והיא מאפשרת למעצב התכניות לספק למשתמש שתי רמות של הודעות שגיאה ואזהרה. דוגמה פשוטה יכולה להסביר את התועלת שבפונקציה זו.

נניח שהמשתמש התקין את התכנית XYZ123A שהיא מערכת הוצאה לאור שולחנית. המשתמש יודע שהיישום דורש דיסק קשיח נוסף, אבל הוא אינו יודע מדוע. במערכת של המשתמש מותקן דיסק קשיח נוסף, ולכן נראה לו שאין בעיה. כאשר המשתמש עמד לשמור את המסמך הראשון שלו כדי לבחון את היישום, הוא קיבל את ההודעה הבאה:

XYZ0000: Warning disk D: is about to be erased!

יישום הדוגמה שלנו מנצל את שירותי ה-Help Message, ולכן המשתמש יכול להתאושש ממצב זה מבלי לפנות למדריך למשתמש. הוא יוכל אז להקיש:

HLPMSG XYZ0000

בנקודה זו, תכנית השירות תשתמש במספר המזהה של ההודעה, כדי לשלוף את הודעת העזרה מקובץ הודעות העזרה של היישום. היא מציגה את ההודעה המקורית ולאחריה את הודעת העזרה. הודעות השגיאה יכולות להראות כך:

XYZ0000: Warning disk %1 is about to be erased!

Explanation: XYZ123A uses a special file system to store its large composite documents with images. XYZ123A is about to initialize drive D: so that it can store these documents.

Action: Copy any existing files on drive D: to another drive before continuing.

שים לב, שניתן להשתמש במשפט HELP לשליפת הודעות לעזרה. משפט זה מאפשר להדליק או לכבות את שורת העזרה במסך, וניתן להשתמש בו גם להפעלת HLPMSG לצורך שליפת הודעות עזרה.

קבצי הודעות העזרה נבנים באותה דרך שבה נבנים קבצי הודעות רגילים. ההבדלים היחידים הם בכך שתו סיווג ההודעות בהודעות העזרה הוא H, והתו האחרון בשם הקובץ הוא "h" כדי לציין שזהו קובץ הודעות עזרה. בדוגמה שלנו שם קובץ ההודעות יהיה:

XYZH.MSG

דוגמה זו מראה גם כיצד מכניסים תוכן משתנה להודעות. אם נתבונן שוב בתרשים 73, נראה שההודעה XYZ0000 אינה מכילה למעשה את אות הכונן. במקומה ישנו הסימן %1, המזהה תוכן משתנה, אשר היישום מחליף באות הכונן המתאימה, לפני שההודעה מוצגת על המסך. זו דוגמה פשוטה המראה איך מספר קטן של הודעות מקוצרות יכול להתאים למגוון רחב של הודעות ספציפיות למשתמש.

ועכשיו נסתכל בשלוש הפונקציות הדינמיות, התומכות באמצעים לטיפול בהודעות של OS/2.

DosGetMessage מאפשרת ליישום לשלוף הודעות שתכנית השירות MKMMSG עיבדה אותן קודם. הפונקציה מחפשת את ההודעה בזיכרון תחילה, ועוברת לחפש בקובץ ההודעות, רק אם ההודעה אינה נמצאת בסגמנט ההודעות הקשור לתכנית. זאת אומרת, שעל-ידי שימוש באמצעים לטיפול בהודעות, יוכל המתכנת לבחור את ההודעות שיוטענו עם התכנית ואת אלו שתשארנה בדיסק. ניתן לעשות זאת מבלי לשנות את קוד המקור של התכנית.

בקריאה לפונקציה זו מעבירים את הפרמטרים הבאים:

- שם קובץ ההודעות (כולל המסלול).
- מספר ההודעה.
- טבלה של מחרוזות להצבה במקום המזהים לתוכן המשתנה, אם הם קיימים.
- מחוונים למאגר, שבו הפונקציה תציב את ההודעה המתקבלת ואת אורכה.

DosInsMessage מבצעת רק את החלפת המשתנים בפונקציה Get Message שתוארה קודם. פונקציה זו שימושית מאוד, מכיון שהיא מאפשרת לתכנית לשלוף הודעה מקובץ הודעות בעזרת הפונקציה Get Message לפני שהתוכן של המשתנים ידוע. כאשר החלק המשתנה של ההודעה ידוע, משתמשים בפונקציה DosInsMessage כדי לבנות את ההודעה המושלמת.

פונקציה זו מספקת אלטרנטיבה לקשירת הודעה למודול טעינה. מעצב תכניות יכול להמנע מהקצאת זיכרון להודעה זו בזמן הביצוע של התכנית, ואז הקצאת זיכרון תתרחש כאשר ההודעה תקשר למודול הטעינה. ניתן לשלוף את ההודעה לפני שהחלק המשתנה שלה ידוע, אבל כל עוד ניתן להכנס לקובץ ההודעות. השליפה עשויה להתבצע לפני פעילות שעלולה למנוע את הגישה לקובץ ההודעות. ניתן להשתמש אז ב-DosInsMessage כדי להוסיף את התוכן המשתנה להודעה לצורך תצוגה.

הפרמטרים המועברים לפונקציה זו הם בהכרח זהים לאלה של DosGetMessage,

פרט לכך שהם כוללים מחוון להודעה שנמצאת כבר בזיכרון, במקום שם קובץ ההודעות.

DosPutMessage משמשת לביצוע פלט של הודעות. הפלט מבוסס על מקשר, ועל כן היישום יכול לבצע את הפלט להתקן פלט קריא, כמו המסך והמדפסת, או שהוא יוכל לבצע פלט לקובץ. מאחר ופונקציה זו מבוססת על מקשר, ניתן לבצע ניתוב ק/פ של ההודעות.

הפרמטרים לפונקציה זו הם קובץ הפלט או המקשר להתקן, מחוון למאגר המכיל את ההודעה ואורך ההודעה.

סיכום של האמצעים לטיפול בהודעות

אנו יכולים לראות עכשיו של-OS/2 יש מערכת מקיפה מאוד של אמצעים לטיפול בהודעות. כפי שציינו קודם, אמצעים אלה הם התשובה לדרישה לספק סביבת מערכת שבה ניתן לתרגם יישומים בקלות לשפות שונות, אך היתרונות שהם מספקים הם רבים יותר. באמצעים אלה לטיפול בהודעות יש גמישות שמאפשרת להם לשפר את הפיתוח, וכפועל יוצא מכך - את השימושיות של היישום. מומחים להנדסת אנוש יוכלו לערוך את תוכן ההודעות מבלי לטפל בתכנות של היישום. המתכנת יוכל לשנות בקלות את המיקום של ההודעות בזמן ריצה כדי לשפר את ביצועי המערכת, והוא יוכל להמנע ממצבים ללא מוצא (כמו ניסיון לקרוא הודעת שגיאה מדיסקט, כדי להודיע למשתמש שהדיסקט אינו ניתן לקריאה). ולבסוף, ניתן לשפר את השימושיות של היישום על-ידי הוספה של רמה נוספת של הודעות עזרה, אשר ניתן לנתב אותן להתקנים וקבצים אחרים.

קביעת התצורה של המערכת למדינות שונות

למוצר בינלאומי כמו OS/2 יש אתגרים טכניים שאיתם הוא צריך להתמודד. הוא צריך להתמודד עם העובדה שבמספר מדינות, יש הבדלים בין המקלדות, המסכים והמדפסות. מדינות שונות מציגות את אותו מידע בצורות שונות, כמו לדוגמה הצגה של התאריך והזמן בצורה שונה, שימוש בסימני מטבע שונים, או שימוש בסימן אחר במקום נקודה עשרונית. כדי לסבך את הדברים, שפות במספר מדינות דורשות סימנים שאין להם כל משמעות בשפה של מדינה אחרת. ואם זה לא מספיק, אז יש מדינות המשתמשות בקודים אחרים בני 8 סיביות כדי לייצג את אותו תו! ולבסוף, מספר מדינות דורשות יותר מ-256 סימנים (תווים), ומכאן שלא ניתן לקדד את כל הסימנים בקוד בן 8 סיביות. הגדרה מפורטת של כל הצירופים בשמונה סיביות נקראת **דף קוד** (code page).

בגירסאות מוקדמות יותר של DOS ניתן למצוא אמצעים המטפלים בחלק מהבעיות הללו. OS/2 מתבססת על הגישה הזו ואף מרחיבה אותה, מכיון שהיא עומדת בדרישות של ציבור בינלאומי גדול הרבה יותר. ציבור גדול הינו פוטנציאל שיווקי גדול יותר ליישומי OS/2.

האמצעים ב-OS/2 שמתמודדים עם האתגרים האלה נקראים באופן כללי התמיכה בשפות לאומיות (National Language Support - NLS). ניתן לחלק את האפשרויות ש-NLS מספקת לשלושה תחומים:

1. התאריך, הזמן וסימן המטבע לפי הנהוג בכל מדינה (נקרא גם פרופיל המדינה).
2. תמיכה במספר דפי קוד.
3. ערכת תווים המבוססת על בית כפול (- Double Byte Character Set DBCS), אשר מאפשרת שימוש בשפות עם יותר מ-256 תווים (למשל, יפנית).

מידע התלוי במדינה

כחלק מארכיטקטורת NLS, כל מדינה מזוהה על-ידי קוד מדינה בן שלוש ספרות. OS/2 משתמשת בפרופיל המידע התלוי במדינה כדי לציין את הדרך שבה כל מדינה משתמשת כדי להציג מידע מסוים. הטבלה המציגה את כל שבע עשרה המדינות שלגביהן יש ב-OS/2 פרופיל מידע נמצאת בתרשים 75. פריטי המידע הנמצאים בפרופיל המדינה הם:

- צורת העריכה של התאריך (שנה, חודש, יום).
- החוצץ בין מרכיבי התאריך.
- צורת העריכה של הזמן (שעות, דקות, שניות).
- החוצץ בין מרכיבי הזמן.
- סימן המטבע.
- צורת הכתיבה של ערכים כספיים.
- מספר המקומות אחרי הנקודה בכתיבת ערכים כספיים.
- החוצץ בין השלם לשבר במספר עשרוני.
- החוצץ במספרים מעל אלף (1,000 - כאן, הפסיק הוא החוצץ).
- החוצץ בין נתונים ברשימה.
- דף קוד עיקרי.

כפי שניתן לראות, המידע התלוי במדינה עונה על תהליכי עיבוד במערכת שחייבים להעשות בצורה שונה בשפות שונות ובמדינות שונות. תהליכי העיבוד של המידע התלוי במדינה משלימים את תכנית התרגום של OS/2 עבור אחת עשרה שפות.

המעבר מדף קוד אחד למשנהו

דף קוד הינו הקשר בין 256 הצירופים במספר בינארי בן 8 סיביות לבין 256 סימנים גרפיים שונים שהם מייצגים. לדוגמה, בדף קוד 437 המספר הבינארי 01000001 מייצג את האות האנגלית A. כפי שהזכרנו בפרק 1, OS/2 תומכת בדף קוד אמריקאי (437, ASCII מורחב), בדף קוד פורטוגזי (860), בדף קוד קנדי-צרפתי (863), בדף קוד נורדי (865) ובדף קוד רב-שפות.

המטרה בהגדרה של דף קוד רב-שפות היא להוציא את כל התווים הגרפיים שאינם הכרחיים, ולהגדיר את 256 הסימנים החשובים ביותר הדרושים לקידוד הטקסט במספר גדול ככל האפשר של שפות. על-ידי שימוש בדף קוד רב-שפות, היישום יכול להגדיל למקסימום את יכולת הניידות שלו בין מדינות שונות.

הצורך בדף קוד אינו מוגבל רק לתרגום שפות. התקני תווים מהדור החדש יכולים לתמוך בדפי קוד מתוכנתים ובהגדרות של סימנים (הנקראים גם טבלת סימנים וטבלת פונטים). תמיכה בהתקנים אלה הינה דרישה נוספת המוטלת על מערכת ההפעלה.

המדינה	קוד המדינה	מקלדת	דף קוד עיקרי	דף קוד משני
ארצות הברית	001	US	437	850
אנגליה	044	UK	437	850
צרפת	033	FR	437	850
גרמניה	049	GR	437	850
איטליה	039	IT	437	850
ספרד	034	SP	437	850
דנמרק	045	DK	865	850
פינלנד	358	SU	437	850
הולנד	031	NL	437	850
נורווגיה	047	NO	865	850
פורטוגל	351	PO	860	850
שוודיה	046	SV	437	850
אסיה	099	--	437	850
אוסטרליה	061	--	437	850
בלגיה	032	BE	437	850
קנדה	002	CF	863	850
אמריקה הלטינית	003	LA	437	850
שוויצריה	041	SF,SG	437	850
ערבית	785	--	864	850
עברית	972	--	862	850
יפן	081	--	932	437
קוריאה	082	--	934	437
סין	086	--	936	437
טיוואן	088	--	938	437

תרשים 75. תמיכת NLS - סימולים של מדינות, מקלדות ודפי קוד

OS/2 מטפלת בדרישות אלה על-ידי כך שהיא עורכת מעקב של דף הקוד הפעיל ביחס לכל תהליך במערכת. וכזכור, ניתן להכין רק שני דפי קוד שונים במערכת. קביעת דפי הקוד נעשית בקובץ CONFIG.SYS וההכנה שלהם מתבצעת בזמן העיבוד של הקובץ הזה בזמן התיחול של המערכת. מערכת ההפעלה תעביר בצורה אוטומטית את דף הקוד המתאים לתהליך הפעיל באותו רגע. דף זה ישמש מעתה לתמיכה בק/פ להתקנים כמו המקלדת, המסך והמדפסת.

דף הקוד העיקרי הוא אחד מהמאפיינים הנכללים במידע המוגדר לכל קוד מדינה. בתרשים 75 תמצא רשימה של מדינות שב-OS/2 יש מידע לגביהן. הרשימה כוללת את קוד המדינה, סידור המקלדת ודף הקוד הראשי והמשני.

ערכת תווים כפולי בתים

ערכת תווים כפולי בתים (Double Byte Character Set - DBCS) מורכבת למעשה מתווים בגודל של בית אחד ומתווים בגודל של שני בתים. טווח הערכים הבינאריים בני 8 סיביות מ-00H עד FFH מחולק על-ידי DBCS לטווחי משנה. אם הקוד של התו נמצא מחוץ לטווחי המשנה, הוא מוגדר כתו בגודל של

בית אחד. אם התו נמצא בתוך אחד מטווחי המשנה, הוא מוגדר כבית הראשון של תו בגודל של שני בתים. ב-OS/2 הטווח שבו ניתן להגדיר טווחי משנה לבית הראשון של תו בן שני בתים מוגבל ל-81H עד FCH (כולל). DosGetDBCSEv הינה פונקציה דינמית שניתן להשתמש בה כדי לקבל את טווחי הערכים של DBCS שצוין, או של ברירת המחדל של המערכת.

המשפט האומר ש-OS/2 מאפשרת DBCS פירושו, שכאשר מרכיבים של OS/2 מטפלים במחרוזות תווים, תווי DBCS לא יפוצלו אלא יוצגו בצורה הנכונה. תכנית התרגום של OS/2 אינה כוללת בגירסה 1.0 שפות DBCS, אבל יש לה את הבסיס לכלול זאת.

האמצעים של OS/2 המטפלים ב-NLS

ניתן לחלק את האמצעים של OS/2 המטפלים ב-NLS לשלוש קבוצות:

1. הוראות התיחול של המערכת הנכללים בקובץ ה-CONFIG.SYS.
2. פקודות שהמשתמש יכול להפעיל במהלך העבודה הרגיל של המערכת.
3. פונקציות דינמיות שהיישום יכול להפעיל.

פקודות NLS בקובץ CONFIG.SYS

בפרק 9 נתאר את קובץ CONFIG.SYS. שלושת המשפטים המוגדרים בקובץ CONFIG.SYS שמתייחסים לתמיכה ב-NLS הם COUNTRY, CODEPAGE ו-DEVIFINO.

COUNTRY=nnn משמש לציון קוד המדינה. המערכת תשתמש במערכת מוגדרת מראש של פריטי מידע התלויים במדינה והקשורים לאותו קוד.

CODEPAGE=xxx,[yyy] משמש להפעלה של דף קוד אחד, או שניים.

DEVINFO= גורם לפקודות מסוימות להשלח להתקן תו התומך במעבר בין דפי קוד. המידע במשפט =DEVINFO והפקודות הנשלחות להתקן מתייחסים להתקן ספציפי.

פקודות NLS למשתמש

CHCP מופעלת על-ידי המשתמש משורת הפקודה ומשמשת לקביעה (או לשאילתה) של דף הקוד ל-session. תהליכים שיופעלו לאחר מכן ב-session זה ישתמשו בדף הקוד שהמשתמש קבע בפקודה זו. תת-מערכות המקלדת, המסך והמדפסת ישתמשו במידע זה, כאשר הם יבצעו ק/פ בשם התהליך.

שים לב: התמיכה בחילופי קודים בתת המערכת של המדפסת מוכלת בתכנית השירות הקובעת את תור ההדפסה (spooler). לכן, כדי להנות מהיתרונות שבחילופי דפי קוד, יש לדאוג שתכנית התורים תהיה פעילה.

משקי תכנות דינמיים ל-NLS

פונקציות NLS הבאות הן חלק אינטגרלי מהמישק לתכנות יישומים ב-OS/2.

DosSetCp קובעת את דף הקוד לתהליך זה. הפרמטרים היחידים הם המספר המזהה של דף הקוד ומלה שמורה של אפסים, אשר משמשים לקביעה של דפי הקוד של המסך והמקלדת. מערכת הקבצים משתמשת בדף הקוד של התהליך כדי לתחל את הק/פ למדפסת. אבל, יש צורך להטעין את ה-spooler אם רוצים שלמדפסת של דף הקוד תהיה השפעה על המדפסת.

DosGetCp מחזירה את מספר דף הקוד הנוכחי שנקבע לתהליך ואת רשימת דפי הקוד העומדים לרשות המערכת. OS/2 מאפשרת כרגע לשני דפי קוד בלבד להיות פעילים במערכת.

DosGetCtryInfo מקבלת את פרופיל המדינה שצוינה, או את ברירת המחדל של המערכת. הפרטים המוחזרים הם: קוד המדינה, דף הקוד הנוכחי של התהליך, וכל שאר הפרמטרים הנכללים בפרופיל המדינה שפורטו קודם לכן.

DosCaseMap מבצעת מיפוי של גודל האות (באנגלית) במחרוזת בינארית. מיפוי לפי גודל האות פירושו מיפוי של מחרוזת מעורבת באותיות קטנות וגדולות לאותיות גדולות.

DosGetDBCSEv מחזירה את טווחי הערכים של ה-DBCS שצוין, או של ברירת המחדל של המערכת. אלה הם טווחי ערכים של תווים המשמשים כבית הראשון בתווים של בית כפול.



פרק 9

שינוי ברירות המחדל: נקודת המבט של המשתמש

כאשר אתה מפעיל את המחשב האישי שלך לאחר שהתקנת את OS/2 על הדיסק הקשיח, פקודות ה-ROM מבצעות בצורה אוטומטית מספר בדיקות. אם לא מתגלות שגיאות בחומרה, פקודות אלו מתחילות את ה-IPL (Initial Program Load) של מערכת ההפעלה. מערכת ההפעלה מתחלת אז את עצמה, וקובעת על-ידי כך את הסביבה שבה אתה רוצה להריץ את היישומים שלך. הקביעה של הסביבה שבה יתבצעו היישומים נעשית בשלושה שלבים:

- שלב הקונפיגורציה (תצורה).
- שלב הביצוע האוטומטי.
- שלב בחירת התכנית.

בשלב הקונפיגורציה, OS/2 מחפשת קובץ מיוחד המכיל מידע על הפרמטרים שנקבעו לתיחול המערכת. שם הקובץ המיוחד הזה הוא CONFIG.SYS, הזהה לשם הקובץ המשמש את DOS לקביעת הקונפיגורציה של המערכת. מערכת ההפעלה קובעת את האופציות לפי הערכים שצוינו בקובץ הקונפיגורציה, ולגבי אותם פרמטרים שלא צוינו או שצוינו בצורה לא נכונה, היא קובעת ברירות מחדל של ערכים ופעולות. התצורה של המערכת חשובה, מכיון שהיא קובעת איך יישומים יטופלו בזמן שהמחשב פועל. במלים אחרות, לאחר שנקבעה תצורת המערכת לא ניתן לשנות אותה בצורה דינמית. ניתן לשנות את קובץ הקונפיגורציה, אבל השינויים ישפיעו רק לאחר שיתבצע אתחול (reset) של המערכת (בעזרת צירוף המקשים Ctrl-Alt-Del המתחיל מחדש את תהליך התיחול), או לאחר כיבוי והדלקה.

לאחר שנקבעה התצורה של המערכת, יישומים יכולים להתבצע ואפשר לעבור לשלב הבא, שלב הביצוע האוטומטי. בשלב זה מערכת ההפעלה מחפשת קובץ מיוחד המכיל פקודות שמשתמשים בהן במנחה הפקודה (command prompt או, שורת הפקודה). קובץ זה נקרא STARTUP.CMD, אשר דומה לקובץ האצווה AUTOEXEC.BAT המשמש ב-DOS. כאשר מערכת ההפעלה מוצאת את קובץ STARTUP.CMD, מעבד הפקודות מתחיל בצורה אוטומטית session של OS/2 ומטפל בפקודות הנמצאות בקובץ זה. כך ניתן להתחיל יישומים בצורה אוטומטית (ב-sessions נפרדים מזה הנוכחי, שבו נמצא מעבד הפקודות) ללא התערבות המפעיל. כל עוד אין פקודה ב-STARTUP.CMD המורה לעבור מ-session זה, ימשיך המשתמש לראות את ה-session של מעבד הפקודות. אם קיימת פקודה המורה לצאת מה-session, מעבד הפקודות יפסיק את פעולתו ולמשתמש יוצג בורר התכניות עם התפריט שלו.

בורר התכניות הוא שלב בחירת התכנית, אשר זמין עד אשר מכבים את המחשב, או מאתחלים אותו. בורר התכניות הוא יישום מיוחד המציג למשתמש במחשב

מישק מבוקר תפריט. המישק הזה נותן את האמצעים לבצע ולשלוט בפעילויות רבות (בתרשים 76 תראה ציור של תפריט בורר התכניות כפי שהוא מוצג לפני שהמפעיל הוסיף יישומים, או התחיל sessions אחרים).

בעזרת בורר התכניות ניתן לבצע את הדברים הבאים:

- להכנס למנחה הפקודות של OS/2.
- להכנס למנחה הפקודות של DOS.
- להתחיל יישומים.
- לעבור בין יישומים.
- לקבל עזרה מקוונת.

כאשר בוחרים במנחה הפקודה של OS/2, המערכת בודקת אם קיים הקובץ OS2INIT.CMD, שמכיל פקודות שניתן להשתמש בהן, בדיוק כמו בקובץ אצווה רגיל. קובץ זה מאפשר לבצע פקודות שיקבעו פרטים מסוימים בסביבה של היישום, כמו מסלולי החיפוש במדריך ליישומים ולקבצי הנתונים שיתבצעו ב-session שהתחיל באותו רגע. כמובן, שאם בוחרים במנחה הפקודות של DOS, מערכת ההפעלה תבצע את קובץ ה-DOS AUTOEXEC.BAT במטרה לקבוע פרמטרים לסביבת היישומים של DOS.

אופציות קונפיגורציה

OS/2 מספקת תכנית התקנה שמסייעת בהכנת המחשב האישי להרצת OS/2. תכנית התקנה זו יוצרת את קובץ הקונפיגורציה, ומאפשרת לבחור אחד מאלה:

- ערכים שנקבעו מראש.
- ברירות המחדל של המערכת.
- ערכים שהמשתמש מכניס לאופציות השונות.

```
Update                                  F1=Help

                 Program Selector

To select a program, press ←, →, ↑, or ↓. Then, press Enter.
To select Update, press F10. Then, press Enter.

Start a Program                  Switch to Running Program
■ Introducing OS/2               ■ DOS Command Prompt
■ OS/2 Command Prompt
```

תרשים 76. התפריט של בורר התכניות ב-OS/2, גירסה 1.0

לאחר ש-OS/2 הותקנה, ניתן לשנות את קובץ CONFIG.SYS באמצעות כל עורך טקסט שהוא. בהמשך נדון באופציות השונות הקובעות איך OS/2 תטפל בסביבת המערכת והיישומים, ונעשה זאת על-ידי הצגת ברירות המחדל של המערכת ומשמעותן למשתמש. היכן שהדבר משמעותי, אנו גם נוקבים בערכים המדויקים שנבחרו על-ידי תכנית ההתקנה. יכול להיות שלא תרצה להשתמש בערכי ברירות המחדל של המערכת ושל תכנית ההתקנה עבור אופציות מסוימות. במקרה זה אתה חייב לכלול את האופציה ב-CONFIG.SYS ולציין את הערך או את הפעולה שאתה רוצה.

תיחול הידברותי

התיחול של OS/2 יכול להיות הידברותי (interactive), או לא-הידברותי.

PAUSEONERROR ברירת המחדל של המערכת: המתן לתגובת המשתמש (על-ידי הוצאת מנחה) לפני שתמשיך את תיחול המערכת.

בזמן ש-OS/2 קובעת את התצורה של עצמה, היא עלולה להתקל בבעיה עם אופציה שצוינה בקובץ הקונפיגורציה. כאשר היא נתקלת בבעיה, היא משתמשת בערך של ברירת המחדל לאופציה זו, כדי לסיים את התיחול. ואז, לאחר שטיפלה באופציות של הקונפיגורציה אבל לפני שהיא מטפלת בביצוע אוטומטי, OS/2 תציג את הודעות השגיאה אם יש כאלו. בנקודה זו, אם בחרת בתיחול הידברותי, מערכת ההפעלה תנחה אותך ללחוץ על מקש Enter כדי לציין שאתה מוכן להמשיך. אם בחרת אחרת, מערכת ההפעלה תעבור אוטומטית לשלב הבא של התיחול. אם אין בעיה באופציות של הקונפיגורציה, OS/2 ממשיכה לשלב הבא מבלי לעצור.

בדרך זו, שבה מערכת ההפעלה ממתינה לתגובה שלך, היא מבטיחה שאתה ער לבעיות שהיא נתקלה בהן עם האופציות של הקונפיגורציה. עם זאת, ברור שמערכת ההפעלה חייבת לתחל את עצמה מבלי להעצר בגלל תקלות, כדי שתוכל לתקן את השגיאות על-ידי שינוי של קובץ הקונפיגורציה, או כדי שתוכל לבדוק את החומרה.

ישנם מצבים שבהם לא יהיה מפעיל ליד המחשב, כמו במקרה של מחשב הממלא תפקיד של שרת קבצים ברשת. במקרה זה, תרצה שמערכת ההפעלה תבצע IPL ללא התערבות המפעיל.

פעילויות של מערכת הקבצים

האופציות המתייחסות למערכת הקבצים משפיעות על ביצועי הק/פ לקבצים בלבד. הביצוע של הק/פ לאובייקטים אחרים שאליהם מערכת הקבצים יכולה להכנס, כמו התקני קבצים למשל, נשאר ללא שינוי.

BUFFERS ברירת המחדל של המערכת: מערכת הקבצים תקצה ותשתמש ב-3 מאגרים (buffers) לניהול הק/פ לקבצים.

ברירת המחדל של תכנית התקנה: מערכת הקבצים תקצה ותשתמש ב-30 מאגרים לניהול הק/פ לקבצים.

מערכת הקבצים משתמשת במאגרים פנימיים כדי לנהל את תנועת נתוני הקובץ בין היישום לבין הדיסק/דיסקט. היישום מציין נתוני קובץ במונחים של בתים, ומערכת הקבצים מתרגמת את הבתים לסקטורים לצורך ביצוע הק/פ להתקן הבלוקים. סקטור הוא בלוק של נתונים בעל אורך קבוע של 512 בתים. נתונים בקובץ ממלאים בד"כ מספר סקטורים בצורה מלאה, והיתרה ממלאת סקטור רק בצורה חלקית (בהתחלה או בסוף הקובץ). מערכת הקבצים משתמשת במאגרים הפנימיים כדי לנהל את הסקטורים החלקיים הללו.

הגדלה של מספר המאגרים מאפשרת למערכת הקבצים לטפל בק/פ ביעילות גבוהה יותר. אולם, ישנה נקודה שמעבר לה הגדלה נוספת במספר המאגרים אינה מביאה לביצועי ק/פ טובים יותר. למעשה יכול לקרות אפילו ההיפך. המספר המתאים ביותר למאגרים במערכת הקבצים תלוי במספר גורמים, כמו כמות הזיכרון שיש במערכת וסוג היישומים שאתה מריץ. כדי לקבוע מה המספר האופטימלי של המאגרים יש לבצע מספר ניסויים עם מספר שונה של מאגרים.

DISKCACHE ברירת המחדל של המערכת: אל תקצה זיכרון מטמון (cache) עבור הדיסק.

ברירת המחדל של תכנית בהתקנה: הקצה 64KB לזיכרון מטמון לדיסק.

ה-device driver של הדיסק הקשיח ב-OS/2 תומך בזיכרון מטמון לגבי אותם דגמים במשפחת PS/2 שעליהם OS/2 יכולה לרוץ (דגמים 50, 60, 70 ו-80). זיכרון מטמון הינו מאגר (buffer) שה-device driver משתמש בו כדי לאחסן באופן זמני מידע שנקרא מהדיסק הקשיח. טכניקה זו עשויה לשפר את ביצועי הק/פ לקבצים. אם המידע הנדרש נמצא בזיכרון מטמון, נחסך הזמן הדרוש להבאתו מהדיסק הקשיח.

כמות הזיכרון האופטימלית שיש להקצות לזיכרון מטמון תלויה במספר גורמים, כמו כמות הזיכרון במערכת, ותדירות ביצוע פעולות הק/פ לקובץ והיקפן. כדי לקבוע את הגודל האופטימלי של זיכרון מטמון יש לערוך מספר ניסויים עם ערכים שונים.

תמיכה בהתקנים

כדי להשתמש בהתקן אחר מאלה שמערכת ההפעלה תומכת בהם בצורה אוטומטית, צריך לספק device driver לאותו התקן. הדבר חשוב במיוחד בהתקן מבוקר פסקים. אופציית התמיכה בהתקן מאפשרת לגרום למערכת ההפעלה להתקין את ה-device drivers לאותם התקנים.

DEVICE ברירת המחדל של המערכת: התקן רק את ה-device drivers הבסיסיים של OS/2.

מערכת ההפעלה מתקינה בצורה אוטומטית device drivers לקבוצה של התקנים הכוללת את המסך, המקלדת, המדפסת והדיסק/דיסקט. הדבר מאפשר שימוש מיידי במספר התקנים. אולם, אתה יכול להשתמש בהתקנים אחרים על-ידי כך שתזהה את ה-device drivers למערכת ההפעלה. אתה יכול גם לציין device drivers שמחליפים device drivers מגירסה קודמת במטרה להחליף אותם, או להרחיב את מגוון הפונקציות שהם תומכים ביחס להתקן. למעשה, אתה יכול גם להתקין device drivers של DOS ולהשתמש בהם בסביבת DOS.

התמיכה במדינות שונות

הדרך שבה מטופלים נתוני המסך, המקלדת והדפסת תלויה בתרבות ובשפה של המשתמש במערכת. מאחר ו-OS/2 הינו מוצר בינלאומי, חשוב שתהיה למערכת זו האפשרות להתאים את עצמה למנהגי המדינה ולשפתה. הפריטים שתלויים במדינה כוללים את צורת ההצגה של התאריך והזמן, חוצץ עשרוני וסימני מטבע. השפה משפיעה על מערכת התווים (דף קוד), מיפוי גודל האותיות, סדרי המיון ועל סביבה של מערכת תווים כפולי בתים.

בפרק 8, בחלק על המידע התלוי במדינה, תמצא תיאור מפורט יותר של האופציות האלה ואיך להשתמש בהן. באופן כללי, בכל פעם שאתה מציין CODEPAGE, אתה חייב לציין גם את DEVINFO.

CODEPAGE ברירת המחדל של המערכת: השתמש במערכת התווים המקומית לגבי המסך והמדפסת, בתנאי שההתקן תומך בפונטים המתאימים. עבור המקלדת השתמש במערכת התווים, כפי שצוינה באופציה COUNTRY.

COUNTRY ברירת המחדל של המערכת: טפל במידע התלוי במדינה לפי הנהוג בארצות-הברית ובשפה האנגלית.

DEVINFO ברירת המחדל של המערכת: אל תכין את ההתקן להחלפת דפי קוד.

היקף ההגנה

המעבד 80286 מספק מנגנון הגנה הכולל רמות הרשאה לביצוע הוראות מוגנות, אבל ההגנה מתייחסת רק ליישומי OS/2.

IOPL ברירת המחדל של המערכת: אל תאפשר ליישומי OS/2 להפעיל הוראות ק/פ מוגנות, כלומר אל תאפשר בקרה ישירה של ההתקן.

יישום OS/2 מתבצע לרוב ברמת ההרשאה הנמוכה ביותר, וכדי שהוא יוכל לבצע הוראות ק/פ, עליו לקבל הרשאת ק/פ (IOPL) ממערכת ההפעלה. ל-device driver ב-OS/2 יש תמיד הרשאת ק/פ כדי שיוכל לשלוט בהתקן. פקודות הק/פ המוגנות כוללות את IN ו-OUT, אשר מעבירות פקודות ונתונים ישירות לכניסות של ההתקן ומתוכן. הן גם כוללות את הפקודות CLI ו-STI, אשר מנתקות ומאפשרות בהתאמה פסקי חומרה.

יישום יכול לפגוע ביכולתם של יישומים אחרים להשתמש בהתקן. לכן OS/2 דורשת מהמשתמש לאפשר ליישום לגשת ישירות לחומרה (היישום אינו יכול לשלוט ישירות בהתקן, אלא אם ציינת את האופציה הזו בקובץ הקונפיגורציה).

PROTECTONLY ברירת המחדל של המערכת: קבע את הקונפיגורציה הן לסביבת DOS (מצב אמיתי) והן לסביבת OS/2 (מצב מוגן).

מערכת המוגנת בצורה מלאה היא זו הפועלת ללא סביבת DOS, אלא במצב מוגן בלבד. בסביבת DOS, או במצב אמיתי, יישום DOS יכול לבצע כל פעולה ללא כל הגבלה, כולל גישה ישירה למרחב הכתובות בין 0 ל-640KB וגישה ישירה

להתקן. אפילו אם אתה קובע שסביבת ה-DOS צריכה להיות קטנה מ-640KB, גם אז יכול יישום ה-DOS להכנס לכל מרחב הכתובות שמתחת ל-640KB. באופן דומה, התקנים שחייבים להיות משותפים במערכת רבת משימות הינם פגיעים יותר מיישומי DOS בגלל האופי החד-משימתי שלהם. לכן אפשר לומר, שההגנה שהמערכת מספקת אינה מושלמת כאשר מתבצעות פעולות בזמן אמיתי.

לעומת זאת, אם תחליט להריץ רק יישומי OS/2, אין לך צורך בסביבת DOS. היתרונות שתשיג הם רמה גבוהה יותר של הגנה ומרחב זיכרון גדול יותר שיעמוד לרשות יישומי OS/2, מכיון שמרחב הכתובות מתחת ל-640KB אינו מוקדש יותר לסביבת ה-DOS.

סביבת היישומים ב-OS/2

כדי לעזור לך לנהל את יישומי OS/2, ישנן אופציות רבות הנוגעות לדרך שבה מערכת ההפעלה תנהל קישור דינמי ולדרך שבה אתה תציין תכניות שאתה רוצה להריץ.

LIBPATH ברירת המחדל של המערכת: השתמש במדריך הראשי של ההתקן שממנו הותחלה (IPL) מערכת ההפעלה (הדיסק הקשיח (\:c) או דיסקט (\:a)), כמקום שבו נמצאות הספריות של השגרות המקושרות בצורה דינמית.

ברירת המחדל של תכנית ההתקנה: השתמש במדריכים שמכילים את קבצי המערכת, כפי שנקבעו על-ידי תכנית ההתקנה.

OS/2 חייבת לאתר את הספריות של המישקים הדינמיים שהיישומים משתמשים בהם, מכיון שהיא צריכה להטעין אותן ולפתור את הקישורים ביישומים. החיפוש אחר הספריות הדינמיות נעשה בצורה אוטומטית, והוא מקיף את כל ה-sessions ואת כל היישומים ב-OS/2.

על-ידי ציון תת-מדריכים שונים לחיפוש אחר ספריות, אתה יכול לארגן את הספריות בתת-מדריכים בהתאם לקריטריונים שלך ולשלוט בסדר שבו יבדקו תת-המדריכים האלה.

PROTSHELL ברירת המחדל של המערכת: השתמש במעבד בפקודות CMD.EXE ובבורר התכניות של OS/2 כמישק למשתמש.

ברירת המחדל של תכנית ההתקנה: השתמש בבורר התכניות של OS/2 כמישק למשתמש ובמעבד הפקודות CMD.EXE עם הפרמטר OS2INIT.CMD.

התכנית של בורר התכניות מתחילה לפעול מיד לאחר סיום התיחול של המערכת, והיא אחראית לניהול הגישה שלך ל-sessions, או ליישומים השונים. מעבד הפקודות מטפל במישק פקודת השורה במצב מוגן.
אתה יכול לבחור במישק למשתמש, ו/או מעבד פקודות משלך.

RUN ברירת המחדל של המערכת: אין תכנית מערכת, שיש להריץ אותה ברקע.

ברירת המחדל של תכנית ההתקנה: התחל את התכנית המטפלת בהדפסות (print spooler).

אתה יכול לבצע ברקע תכנית לא-הידברותית במשך התיחול של המערכת, ולפני שתכנית המישק למשתמש תותחל. תכנית זו אינה יכולה לבצע ק/פ למסך, למקלדת ולעכבר, אלא דרך "pop-up".

ריבוי משימות

גורמים רבים במערכת רבת משימות משפיעים על הביצועים. ניתן לכוון אותם ולהתאימם לדרישותיך.

MAXWAIT ברירת המחדל של המערכת: השתמש בערך של שלוש שניות עבור פסק הזמן.

קיימים במערכת threads רבים בסיווג רגיל, בעלי רמות עדיפות ביצוע שונות. כתוצאה מכך, thread יכול להמתין זמן מסוים עד שהוא יתבצע. תופעה זו נקראת לפעמים "הרעבה של היע"מ". כדי למנוע מצב שבו אין גישה ליע"מ, מערכת ההפעלה מגבילה את זמן ההמתנה של thread רגיל. לאחר שזמן ההמתנה עבר, היא מעלה באופן זמני את רמת העדיפות של ה-thread כדי לתת לו סיכוי להתבצע למשך פרק זמן מינימלי.

גורם זה משפיע על הביצוע של פעילויות הנמצאות ברמת עדיפות רגילה. הערך האופטימלי תלוי בגורמים הבאים:

- מספר היישומים שאתה מריץ בו-זמנית.
- סוגי הפעילויות שהיישומים מבצעים.
- התדירות שבה מתבצעות הפעילויות האלו.

במלים אחרות, אתה יכול לנסות ערכים שונים ביחס ליישומים שאתה בוחר להשתמש בהם.

PRIORITY ברירת המחדל של המערכת: התאם בצורה דינמית את רמת העדיפות של thread רגיל.

כדי להגיע לאופטימיזציה בביצוע של threads בסיווג רגיל, מערכת ההפעלה עוקבת אחר סוגים שונים של פעילויות, כמו שימוש ביע"מ, או שימוש בק/פ. לאור זאת היא משנה בצורה דינמית את העדיפות של threads אלה.

THREADS ברירת המחדל של המערכת: קבע שהמספר הכולל של threads שיכולים לרוץ במערכת הוא 64.

רמת הביצועים של יישומים במערכת בעלת תצורה מסוימת תלויה במספר ה-threads שאנו מאפשרים לבצע בה בו-זמנית.

אם תוסיף כרטיסי הרחבה ויישומים למערכת, יתכן ותצטרך לאפשר הפעלה של threads נוספים. אם לא תבצע זאת, תגלה שרמת הביצועים יורדת מכיון שכרטיסי ההרחבה והיישומים נאלצים לבצע את הפעילויות שלהם בצורה סידרתית ולא בו-זמנית.

TIMESLICE ברירת המחדל של המערכת: קבע שפלח-הזמן המקסימלי הוא 248 אלפיות השניה ופלח-הזמן המינימלי הוא 32 אלפיות השניה.

פלח-הזמן המקסימלי מגדיר את פרק הזמן המקסימלי שבו thread יתבצע, לפני שהוא מאפשר ל-thread אחר הנמצא באותה רמת עדיפות להתבצע. חלוקה לפלחי זמן לפי השיטה המחזורית מבטיחה של-threads בעלי רמת עדיפות זהה יהיו סיכויים שווים להתבצע. פלח-הזמן המינימלי משמש לשליטה במצבים מיוחדים.

אם יש יישומים שמבצעים פעילויות התלויות בזמן, תצטרך להקטין את פלח-הזמן המקסימלי, כדי לאפשר להן להתבצע בתדירות גבוהה יותר. אולם, אתה צריך להמנע מערכים נמוכים מדי, כי בנקודה מסוימת מערכת ההפעלה תבזבז יותר זמן על תזמון וניתוב ה-threads מאשר על הביצוע שלהם. אתה צריך להמנע מערכים גדולים מדי, כי אתה יכול לגרום לבזבוז של זמן רב בהמתנה של ה-threads לביצוע.

ניהול זיכרון

אתה יכול לשלוט בגורמים מסוימים המשפיעים על ניהול הזיכרון במערכת שלך.

MEMMAN ברירת המחדל של המערכת: השתמש בהעברה והזזה של סגמנטים, אם התחלת מדיסק קשיח. השתמש רק בהזזת סגמנטים, אם התחלת מדיסקט.

בניהול הקצאת שטחי האחסון בדיסק/דיסקט נמצא שני גורמים המשפיעים על ביצועי המערכת: העברה והזזה של סגמנטים. העברה של סגמנטים מאפשרת למערכת ההפעלה לשמור באופן זמני סגמנטים של נתונים על הדיסק במטרה לשחרר זיכרון פיסי לשימושים אחרים (אין צורך לשמור סגמנטים של פקודות, אלא רק לבטל אותם בזיכרון ולשלוף אותם שוב בעת הצורך מהדיסק). הזזת סגמנטים מאפשרת למערכת ההפעלה להזיז סגמנטים בזיכרון הפיסי כדי לצרף יחד חלקים פנויים בזיכרון לגוש רציף גדול יותר. העברה והזזה של סגמנטים מאפשרים ליישומים לתמוך בזיכרון-יתר. הזזת סגמנטים בלבד מאפשרת שימוש יעיל יותר בזיכרון הפיסי.

אם יש יישום או כרטיס הרחבה שתלוי מאוד בתזמון, תצטרך למנוע העברות והזזות כדי להבטיח את מהירות התגובה הדרושה. אולם, הביצועים יבואו על חשבון ניצול הזיכרון. במלים אחרות, לא תוכל להריץ אותו מספר יישומים כמקודם, וגם היישומים שתריץ יהיו מוגבלים לכמות הזיכרון שבמערכת.

SWAPPATH ברירת המחדל של המערכת: קובץ הסגמנטים המועברים של המערכת נמצא במדריך הראשי שבהתקן המשמש לתיחול (IPL) מערכת ההפעלה - הדיסק הקשיח (\:c) או דיסקט (\:a).

קובץ הסגמנטים המועברים (swap file) משמש לאחסון זמני של סגמנטים של נתונים שהוצאו מהזיכרון כאשר התקבלה בקשה להקצאת זיכרון שאי אפשר היה למלא אותה. אתה יכול לבודד קובץ זה בתת-מדריך או במחיצה נפרדת בדיסק הקשיח. רצוי יותר להשתמש במחיצה, כי הקובץ יכול להגיע לממדים גדולים.

סביבת DOS

אתה יכול לשלוט בגודל של סביבת DOS ב-OS/2.

RMSIZE ברירת המחדל של המערכת: קבע את גודל סביבת DOS לפי כמות הזיכרון הכולל במערכת מינוס 512KB, או לפי הכתובת הגבוהה ביותר הנמצאת מתחת ל-640KB (כלומר, 512KB או 640KB).

במונחים כלליים, OS/2 מעמידה לרשות סביבת DOS את כמות הזיכרון הגדולה ביותר האפשרית מתחת ל-640KB. בחישוב זה היא לוקחת בחשבון את הזיכרון הדרוש לפעילויות במצב מוגן. אולם, אם אתה לא צריך כמות כזו של זיכרון ליישומי DOS שאתה מריץ, אתה יכול לציין ערך קטן יותר. כל כמות זיכרון שלא תשתמש בה עבור יישומי DOS תשמש לפעילויות במצב מוגן.

OS/2 תומכת במספר אופציות ש-DOS משתמשת בהן: BREAK, FCBS ו-SHELL. אופציות אלו משפיעות על סביבת DOS בלבד. ברירות המחדל ש-OS/2 קובעת לאופציות אלו הן כדלהלן:

BREAK ברירת המחדל של המערכת: אל תבדוק את צירוף המקשים Ctrl-Break בסביבת DOS, פרט למקרים בהם מתבצעות פעולות סטנדרטיות בהתקן.

FCBS ברירת המחדל של המערכת: אפשר ל-16 גושי בקרה של קבצים (FCB) להפתח בו-זמנית. אל תאפשר שימוש חוזר ב-8 הראשונים, כאשר יש צורך ביותר מ-16 FCBs.

SHELL ברירת המחדל של המערכת: השתמש במעבד הפקודות של OS/2 התואם את סביבת ה-DOS שהוא COMMAND.COM.

ביצוע אוטומטי

OS/2 מספקת אמצעי לתיחול אוטומטי בקובץ האצווה STARTUP.CMD הדומה מבחינות רבות לקובץ AUTOEXEC.BAT ב-DOS. אם קובץ STARTUP.CMD נמצא במערכת, OS/2 תתחיל את ה-session הראשון עם מעבד הפקודות של OS/2 כדי שיתרגם את הקובץ. כאשר אתה משתמש במנגנון זה אינך חייב לעבור דרך המישק למשתמש כדי להריץ יישומים. אם לא ניתן למצוא את קובץ STARTUP.CMD, מערכת ההפעלה מריצה את התכנית המשמשת כמישק למשתמש. בתרשים 77 תמצא דוגמה של קובץ STARTUP.CMD.

פקודת המפתח בקובץ STARTUP.CMD היא הפקודה START. אתה יכול להשתמש בפקודה זו במנחה הפקודות של OS/2, אך היא יעילה הרבה יותר אם נשתמש בה בשיתוף עם מנגנון התיחול האוטומטי. תכנית או קובץ אצווה המזוהים על-ידי פקודת START מוכנסים לתוך session נפרד ומבוצעים בו. ישנה פקודה דומה, הנקראת RUN, שמשתמשים בה בקובץ ה-CONFIG.SYS. אולם, ניתן להשתמש בפקודת RUN רק להרצת תכניות לא-הידברותיות, אשר מתבצעות בעיקר ברקע.

ציון פרמטרים של הסביבה באמצעות הפקודות PATH, DPATH, PROMPT ו-SET משפיע רק על היישומים שפועלים באותו session שבו מתבצע הקובץ STARTUP.CMD. אפשר לגרום לקובץ אצווה אחר להתבצע ב-session אחר על-ידי שימוש בפקודת START, ואז הוא יוכל לבצע פקודות שיקבעו את הסביבה. פקודות אלו תתבצענה לפני שיבוצע היישום.

```
PATH c:\;c:\os2;c:\batch;
START acomm
START editor
START appoint
EXIT
```

1. קובע את מסלול החיפוש במדריך כדי לאתר תכניות.
2. מתחיל session אחר המריץ את תכנית התקשורת האסינכרונית - acomm.
3. מתחיל session אחר המריץ את תכנית העריכה - editor.
4. מתחיל session אחר המריץ את תכנית סידור הפגישות - appoint.
5. מפסיק את פעולת ה-session הנוכחי.

תרשים 77. דוגמה של קובץ האצווה STARTUP.COM

בחירת תכניות

בחירת תכניות הוא השלב שבו מתבצעות כל הפעולות של המפעיל, עד אשר מכבים את המחשב או מאתחלים אותו שוב. ברירת המחדל של OS/2 לתכנית המישק למשתמש היא בורר התכניות (Program Selector). תכנית זו מאפשרת לנהל את היישומים הרצים במערכת באמצעות מישק מבוקר תפריט. כמובן שאתה יכול תמיד לבחור במנחה הפקודות ולהשתמש במנחה הפקודות של OS/2, או של DOS. ממנחה זה תוכל להתחיל יישומים ו-sessions שונים. אם אינך מעוניין להשתמש בתפריט של בורר התכניות כדי לעבור לעבור session, תוכל להשתמש בצירוף מקשים (Alt-Esc) כ-"מקשים חמים", שבעזרתם תוכל לעבור מ-session פעיל אחד למשנהו. אם החלטת בנקודה מסוימת להפעיל את התפריט של בורר התכניות, תוכל להשתמש בצירוף מקשים אחר (Ctrl-Esc), כדי שיעבירו אותך לבורר התכניות.

כל יישום OS/2 שאתה עובר ממנו ממשיך להתבצע ברקע. אולם אם אתה עובר מסביבת DOS, הביצוע בה מושעה עד אשר אתה חוזר אליה.

פקודות וקבצי אצווה

קבצי אצווה משרתים את אותה מטרה כפי שנעשה ב-DOS: הם מאפשרים לבצע בצורה אוטומטית רצף של פקודות ממנחה הפקודות מבלי שתצטרך להקיש את הפקודות בעצמך. פגשנו כבר מספר קבצי אצווה מיוחדים כמו OS2INIT.CMD, STARTUP.CMD ו-AUTOEXEC.BAT.

קבצי האצווה ב-OS/2 מזוהים בעזרת ההרחבה CMD., בניגוד להרחבה BAT., המשמשת את קבצי האצווה של DOS. קובץ אצווה עם ההרחבה BAT. לא יוכל להתבצע בשורת הפקודה של OS/2 וגם להיפך, קובץ אצווה עם הסיומת CMD. לא יוכל להתבצע בשורת הפקודה של DOS. הבדל זה עוזר לך להמנע מלהריץ קובץ

אצווה בסביבה לא נכונה. הדבר חשוב, כי אתה עלול להריץ קובץ אצווה שישנה את הסביבה באופן "קבוע" (על-ידי מחיקת קבצים, למשל). כתוצאה מכך אתה עלול לגלות מאוחר מדי, שהיישום שקובץ האצווה התחיל אותו אינו יכול לרוץ במצב הנוכחי של המעבד, ושאין לך אפשרות להתאושש מהשינויים.

כמו ב-DOS, קובץ אצווה ב-OS/2 לא רק מטפל בפקודות, אלא גם מספק מספר פקודות שניתן לשלוט בעזרתן על הדרך שבה מתבצע קובץ האצווה. פקודות אלו כוללות:

- אפשרות לקרוא לקבצי אצווה אחרים.
- החלפת פרמטרים בפקודות.
- שליטה בתצוגה של פקודות קובץ האצווה.
- הסתעפות לקטעים אחרים של קובץ האצווה.
- לולאה לביצוע חוזר של פקודות.
- בחינה של מצבי שגיאה.

בתרשים 78 תמצא רשימה של פקודות האצווה.

Command	OS/2	DOS
CALL	*	*
ECHO	*	*
ENDLOCAL	*	
EXTPROC	*	
FOR	*	*
GOTO	*	*
IF	*	*
PAUSE	*	*
REM	*	*
SETLOCAL	*	
SHIFT	*	*

תרשים 78. פקודות אצווה

פרק 10

להיכן מתקדמים מכאן?

המהדורה הסטנדרטית של OS/2 מאפשרת ליישומים לנצל בצורה מלאה את הסביבה התפעולית של המיקרו-מעבד 80286, הן במונחים של הגנה וזיכרון פיסי גדול והן במונחים של ביצוע רב משימות וקישור דינמי. זאת באשר ליישומים, "הנהנים" משימוש ב-OS/2. אך גם למשתמש במחשב האישי יתרונות בנושאים הבאים:

- אפשרות להריץ מספר יישומים במקביל.
- תהליך התקנה הידברותי.
- תאימות עם DOS.
- בורר תכניות קל לשימוש.

מפתח היישומים יהנה גם הוא מהארכיטקטורה הפתוחה ומהעזרים הנמצאים בערכת הכלים למתכנת. המהדורה הסטנדרטית של OS/2 הינה הגרעין שישמש בסיס לצמיחה עתידית. המהדורה הסטנדרטית של OS/2 מוצגת (בעת כתיבת הספר) בשתי גירסאות (ראה תרשים 79). גירסה 1.1 משלימה את גרעין מערכת ההפעלה עם מישקים לביצוע חלונות וגרפיקה ועם שיפורים במישק למשתמש המשולבים עם מנהל התצוגה.

השלב הבא ב-OS/2 הוא המהדורה המורחבת - OS/2 Extended Edition - או: OS/2 EE. מהדורה זו מוסיפה שירותים לניהול תקשורת ובסיס נתונים (ראה תרשים 80), והינה פתרון יחיד לאלה הדורשים שירותים אלה. המהדורה המורחבת היא גמישה: אתה יכול לבחור ולהתקין את השירותים שאתה זקוק להם. התמיכה בתקשורת מכסה מגוון רחב של חיבורים ופרוטוקולים, עם הדמייה של מספר רב של מסופים ופרוטוקולים להעברת קבצים. התמיכה בבסיס הנתונים הינה באמצעות המודל של בסיס הנתונים הטבלאי, שבו הניהול של הנתונים מתבצע בעזרת שפת הניהול והשאילתות הידועה בשם SQL.

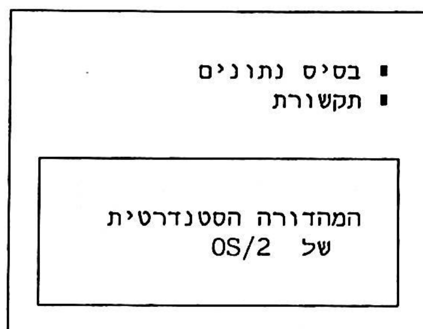
גירסה 1.1

▪ חלונות ▪ גרפיקה
▪ הגנה ▪ זיכרון פיסי גדול ▪ זיכרון בפועל ▪ ביצוע רב משימות ▪ מערכת רבת-תכניות ▪ קישור דינמי ▪ תאימות ל-DOS

גירסה 1

▪ הגנה ▪ זיכרון פיסי גדול ▪ זיכרון בפועל ▪ ביצוע רב משימות ▪ מערכת רבת-תכניות ▪ קישור דינמי ▪ תאימות ל-DOS

תרשים 79. המהדורה הסטנדרטית של OS/2



תרשים 80. המהדורה המורחבת של OS/2

מנהל התצוגה

מנהל התצוגה (Presentation Manager) מספק תמיכה לביצוע חלונות וגרפיקה. חלונות מאפשרים "להסתכל" במספר יישומים בצורה סימולטנית, ויש אפשרות ליישום אחד להיות בעלים של יותר מחלון אחד. אתה יכול לשלוט בגודל החלון ובמיקומו, ליצור ולבטל חלונות. ניתן להעביר נתונים מחלון אחד לחלון אחר, ופירושו של דבר, שאפשר להעביר נתונים מיישום אחד לחברו. הגרפיקה מאפשרת ליישומים לנצל מסכים התומכים במצב שבו כל הנקודות במסך ניתנות למיעון (All Points Adressable - APA). המישקים תומכים בפעולות בווקטורים, בפעולות ב-raster ובמגוון רחב של סוגים וגדלים של פונטים.

מנהל בסיס הנתונים

מנהל בסיס הנתונים (Database Manager) מספק מודל יחסי של נתונים הדומה ל-DB2 ול-SQL/DS של IBM. ההגדרה, השליפה, השינוי והבקרה של הנתונים מתבצעים באמצעות SQL. SQL היא שפה עילית לניהול נתונים, שהן המשתמשים (בדרך הידברותית) והן היישומים (באמצעות תכנות), יכולים להשתמש בה.

הנתונים מאורגנים במבנה של טבלאות פשוטות בעלות רשומות ושדות. אתה מגדיר נתונים ופועל עליהם במונחים של טבלאות, ואינך חייב להיות מומחה במבני נתונים מסובכים כדי לנהל את מאגר הנתונים שלך. השפה SQL מאפשרת לך להצביע על מה שאתה רוצה לעשות, והיא תדאג לביצוע. אתה יכול לבצע פעולות אריתמטיות על נתונים, לאחזר נתונים בצורה סלקטיבית, למיין נתונים בצורה דינמית ולהכין דוחות המותאמים לצרכיך.

מנהל בסיס הנתונים מספק גם פונקציות המשמשות ליבוא/יצוא של נתונים. פונקציות אלו משמשות להחלפת נתונים עם יישומים אחרים המבוססים על בסיסי נתונים אחרים הפועלים על PC.

מנהל התקשורת

מנהל התקשורת (Communications Manager) במהדורה המורחבת משרת הן את משתמש הקצה והן יישומים שצריכים תמיכה בתקשורת. באמצעות מנהל התקשורת ניתן לבצע תקשורת בין מחשבים אישיים, ובין מחשב אישי לבין מחשב מארח.
מנהל התקשורת תומך בסוגים רבים של חיבורים, וביניהם:

- SDLC.
- DFT (Distributed Function Terminal) ל-IBM 3174 או ל-3274.
- רשת טבעת הסמן (Token-Ring) של IBM.
- רשת PC של IBM.
- ערוצים אסינכרוניים.

מנהל התקשורת משתמש בפרוטוקול המתאים לכל קישור: LU6.2, LU2 עבור 3270, פרוטוקולים אסינכרוניים ואחרים. המערכת כוללת תוכנה להדמייה בו-זמנית של סוגים שונים של מסופים כמו IBM 3270, IBM 3101 ומסופי ASCII אחרים. יש תמיכה בהעברת קבצים בתקשורת. כדי לספק יותר גמישות, ניתן לבחור ולהתקין בצורה דינמית את תכניות השירות השונות לפי צרכי המערכת. מנהל התצוגה מספק התראות לניהול הרשת, פונקציות לאיתור בעיות ובקרות שונות אחרות.

מנהל התקשורת משפר את ביצועי היישום בעזרת מספר מישקי תכנות, אשר כוללים:

- תקשורת מתקדמת בין תכנית לתכנית (Advanced Program-to-Program Communication - APPC). ארכיטקטורת LU6.2 מציינת את הפונקציות שיישומים יכולים להשתמש בה לצורך APPC בערוצי נתונים. עומדות לרשותך פונקציות התלויות בזרם הנתונים וכאלו שאינן תלויות בזרם הנתונים.
- מישק לתכנות Server-Requester (Server-Requester Programming Interface - SRPI). יישומים מבקשים יוכלו לקבל שירותים מיישומים מארחים בעזרת הפרוטוקול LU2.
- מישק של תקשורת אסינכרונית (Asynchronous Communication Device Interface - ACDI). יישומים יכולים לנהל נתונים בערוצים אסינכרוניים, ולשלוט במאפיינים של קו התקשורת עם ACDI.

מישקים אחרים במערכת הם: מישק לתכנות 3270, מישק ל-IBM NetBIOS, מישק ל-IEEE 802.2.

קרש קפיצה

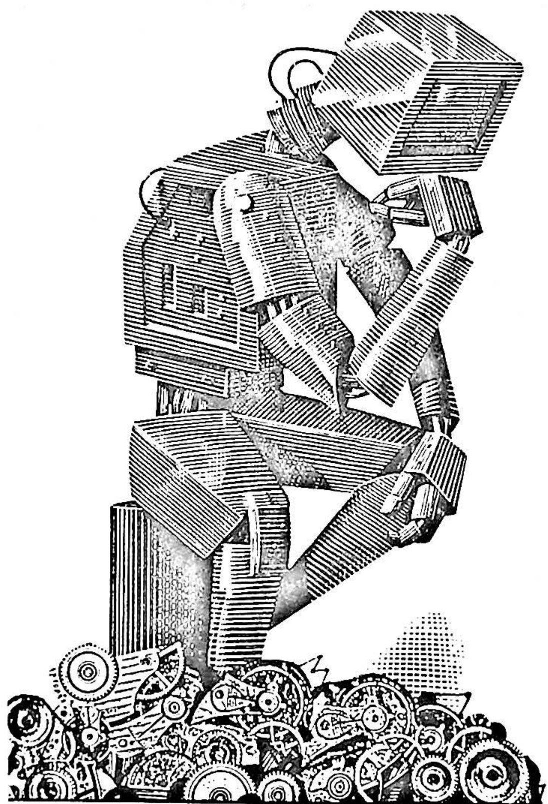
מהדורה 1.0 של OS/2 מאפשרת לך להתחיל את שני השלבים הראשונים בתהליך מעבר רב-שלבי לתחנת העבודה של העתיד. השלבים הראשונים הם כדלהלן:

- כתיבה מחדש של יישומי DOS קיימים, כדי שיוכלו לפעול בסביבת הביצוע של OS/2. לרשותם יעמדו מספר sessions ש-OS/2 מספקת, אך רק session אחד של DOS.

- תכנון של יישומים חדשים, וכתיבה מחדש של יישומים קיימים כדי שינצלו את הפונקציות והמאפיינים החדשים של OS/2. לרשות המתכננים יעמדו התמיכה בזיכרון פיסי גדול וביצוע רב משימות.

השלבים הבאים המתוכננים הם כדלהלן:

- ניצול הגרפיקה וסביבת החלונות שמאפשר מנהל התצוגה ב-OS/2 גירסה 1.1.
- שילוב האמצעים המסופקים על-ידי מנהל בסיס הנתונים ומנהל התקשורת לתוך יישומים חדשים וקיימים.



נספח א
מנהל התצוגה

מנהל התצוגה (Presentation Manager - PM) הינו המישק למשתמש, ב-OS/2 גירסה 1.1 ובגירסה המורחבת. מנהל התצוגה מספק למשתמש מישק גרפי המבוסס על חלונות מסך לשם הצגת פקודות ומאפשר לו לתפעל את המערכת בצורה יעילה יותר לעומת מישק מסורתי המבוסס על שורת פקודה. פרק זה מציג את הגישות הבסיסיות המיושמות על-ידי מנהל התצוגה ומדגים תהליך בנייה של שלד של יישום.

מהו מנהל התצוגה?

התשובה לשאלה זו תלויה בשואל, כלומר האם הוא משתמש הקצה או המתכנת. מנקודת המבט של **המשתמש**, מנהל התצוגה הינו הקליפה (shell) שבעזרתה הוא מפעיל יישומים בצורה הידברותית. מנקודת המבט של **המתכנת**, מנהל התצוגה הינו אוסף של שירותי API נוספים, הקשורים לפילוסופיה כללית לתכנון יישומים. מנקודת מבטו, מנהל התצוגה הינו ערכת כלים של פונקציות רבות הקשורות זו בזו. אם נשתמש בפונקציות אלו באופן נכון, הן תאפשרנה לתכניות יישום שונות להשתמש במישק משותף.

המטרה של מנהל התצוגה היא לאפשר למי שיש לו ידע בסיסי על המערכת להריץ כל יישום ללא הדרכה מוקדמת. כלומר, מי שיודע להריץ תכנית אחת המתאימה למנהל התצוגה, ידע להריץ את כולן. אולם, במציאות יש צורך בהדרכה מסוימת, אשר מתייחסת למה שהתכנית עושה ולא לדרך שבה יש להפעיל אותה.

בנקודה זו חשוב מאד להבין, שלא כל תכנית התואמת למנהל התצוגה מציגה למשתמש מישק הדומה לזה של מנהל התצוגה. אפשר לעקוף את התפישה הבסיסית של מנהל התצוגה, אבל צריכה להיות סיבה טובה לכך, כי הדבר יפריע למשתמשים אחרים בתכניות. מי שמפתח תוכנה ורוצה למכור אותה, צריך לדאוג לכך שהתכניות יתאימו לתפישה הכללית של המישק לתכנות יישומים של מנהל התצוגה, אשר תואם את ארכיטקטורת היישומים הבין מערכתית (SAA).

ערכת כלים למתכנת

ערכת כלים למתכנת (Programmer's Toolkit) הינה אוסף של שגרות וכלי עזר אחרים המיועדים לשרת את המתכנת ב-OS/2 ואת מי שמשתמש במערכת. המגוון מכסה תחומים רבים, אך הוא משרת במיוחד את מנהל התצוגה:

- Include and Header files

אלה הם קבצי כותרת המיועדים למתכנת בשפת C, ותכניות מקור שונות עבור

המתכנתים בשפת אסמבלר. הם כוללים הגדרות והכרזות לשם שילוב עם מנהל התצוגה ומישקי תכנות (API). המשתמש יכול לבחור חלק מן ההגדרות והשגרות בהתאם לצרכים שלו: שירותי מנהל התצוגה, הגדרות של קבועים וסוגי משתנים, הודעות שגיאה, שגרות API ועוד. השימוש בקבצים ובשגרות יוסבר בתכנית לדוגמה.

- Presentation Manager Sample Programs

המערכת כוללת מספר תכניות שלד המדגימות שירותים ושימושים שונים של מנהל התצוגה ואת טכנולוגיית החלונות. ניתן להפעיל תכניות אלו לשם לימוד וכדי להפיק תכניות המותאמות לצרכי היישומים: שרטוט מסך לשם קליטה והפקה של נתונים, תכניות הידברות, גרפיקה, שילוב טכסט וגרפיקה ועוד.

- Presentation Manager tools

קבוצה זו כוללת שגרות שונות של העורכים (editors) הכלולים בערכת מנהל התצוגה, ספריות יבוא שונות, תכניות קישור ועוד.

- Base sample programs

תכניות המדגימות טכניקות תכנות שונות ב-OS/2, בשפת C ובאסמבלר. המשתמש יכול לנצל דוגמאות אלו כדי לפתח תכניות המותאמות לצרכיו.

- Base Message utilities

שגרות שירות לבניית קובץ הודעות שגיאה המופרד מתכנית היישום, אך מקושר אליו בעזרת אינדקס.

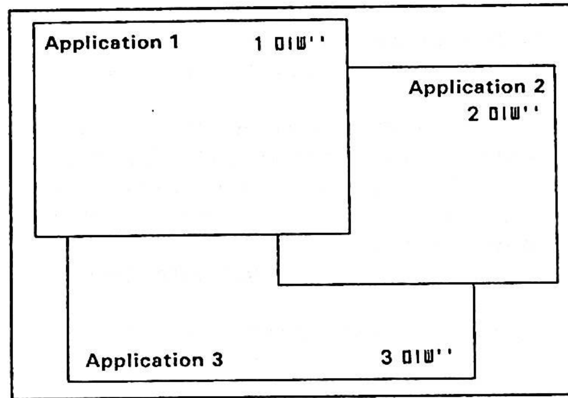
חלונות על פני המסך

חלון (window) הינו צורה מלבנית על פני המסך, אשר מכסה את כולו או את חלקו. יישום בבקרת מנהל התצוגה יכול ליצור חלון אחד או יותר, ולהשתמש בהם כמו בדפי נייר על שולחן העבודה. לעתים כולם גלויים ולעתים כמה מהם מוסתרים, כפי שמוצג בתרשים 1.

החלון העליון החשוף, או אחד מהחלונות הגלויים על פני המסך מייצג את התכנית הפעילה באותו זמן במחשב. המשתמש יכול לבחור כל חלון כדי שיהיה פעיל, כלומר, להציב אותו מעל כל החלונות. ניתן להקטין או להגדיל את שטח החלון הפעיל ולהזיז אותו לכל מקום על פני המסך.

החלונות מסודרים בצורה הירככית. החלון העליון הוא **חלון שולחן העבודה** (desktop window), אשר מקביל למסך השלם. אחר-כך אנו יכולים לראות חלונות אשר מכסים חלק מהמסך ומוסתרים בחלקם, או כולם על-ידי חלונות שנמצאים מעליהם. החלונות ברמה הגבוהה ביותר הם **חלונות ראשיים** (main), או **עליונים** (top level), אשר בעזרתם מזהים את היישום.

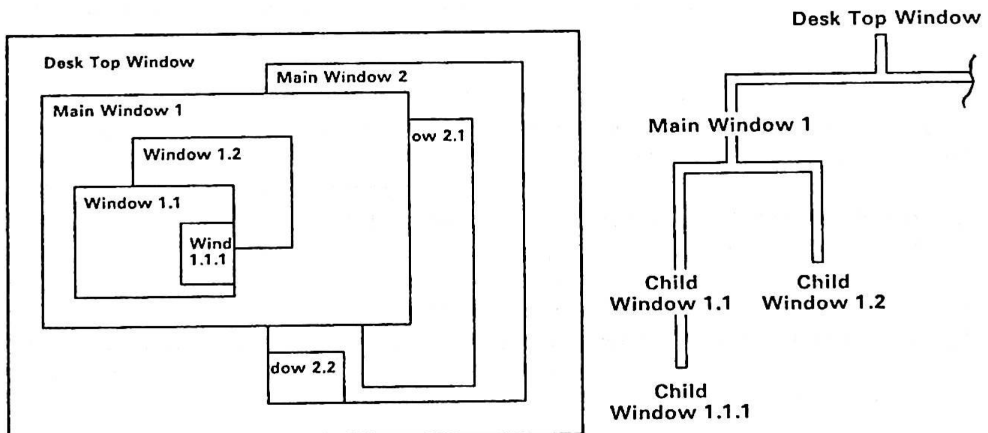
חלון שולחן העבודה הוא חלון אב (parent) וכל חלון ראשי של יישום הוא בן (child). כך גם חלון היישום יכול להיות אב של חלונות בנים הנוצרים תחתיו, ואשר מייצגים פעילויות או נתונים שונים ביישום. הירכיה זו מוצגת בתרשים 2.



תרשים 1. חלונות של יישומים על פני המסך

מנהל התצוגה מספק חלונות למטרות מיוחדות. הנפוצים ביותר מבין אלה הם **התפריט** (menu) **ותיבות ההידברות** (dialog boxes). תפריט הוא חלון שמכיל אפשרויות פעולה הניתנות לבחירת המשתמש. במקום לספק את הפונקציות המאפשרות בחירה מתפריט בתכנית המשתמש, ניתן ליצור חלון תפריט סטנדרטי על-ידי שירותי מנהל התצוגה. תיבת הידברות היא חלון שמאפשר הידברות יותר מורכבת עם היישום מאשר התפריט. מנהל התצוגה משתמש בדרך זו כדי לקבל קלט שאינו מגיע מהתפריט, כמו למשל שם של קובץ שיש להפעיל.

עכבר: העכבר הוא כלי חשוב בתפעול חלונות תצוגה. מנהל התצוגה מאפשר להשתמש בעכבר כמעט לכל פעילות של בקרה, בחירה ושרטוט, הרבה מעבר לאפשרויות שב-DOS ובגירסה המקורית של OS/2. ניתן לקבל קלט מהעכבר כשם שניתן לקבל אותו מהמקלדת. תכנית יישום יכולה להתעלם מהעכבר, אך זה יהיה בניגוד לעקרון בסיסי שעל פיו תוכנן מנהל התצוגה.



תרשים 2. מערך הירכי של חלונות במסך

כדי להפעיל בחירה מסוימת, מזיזים את מחוון העכבר למקום הנבחר ולוחצים פעמיים (double click) ברצף מהיר על הלחצן השמאלי. אפשר להזיז אובייקטים במסך על-ידי הזזת מחוון העכבר אל האובייקט, לחיצה קבועה על הלחצן השמאלי, הזזת מחוון העכבר למקום החדש ושחרור הלחצן.

סמלים וצורות: מנהל התצוגה מאפשר (אין זה חובה) להשתמש **בסמלים** (icons) **ובמיפוי סיביות** (bit-maps) לשם העברת מידע למשתמש. השימוש בסמלים ובצורות גרפיות מוסבר על-ידי פתגם האומר, כי "תמונה שווה אלף מלים". סמל ב-OS/2 הוא סימן קטן שמייצג פונקציה, או תכנית, שניתן להפעיל אותה על-ידי הזזת העכבר לסמל ולחיצה כפולה עליו. **צורה גרפית** (graphic image) משמשת בדרך-כלל להצגת מידע.

בקרת ק/פ: מנהל התצוגה חייב לשלוט בכל הקלט והפלט, ולכן לא ניתן להשתמש ברבות מפונקציות הספריה הסטנדרטיות בשפת C, כמו **printf()** ו-**scanf()**, כאשר התכניות פועלות בבקרת מנהל התצוגה. למעשה, אחת הסיבות שיש למנהל התצוגה שירותים רבים כל כך היא, שהם חייבים להחליף מספר גדול של פונקציות סטנדרטיות בשפת C.

יישום בבקרת מנהל התצוגה

מהלך הביצוע של יישום הפועל תחת מנהל התצוגה שונה באופן בסיסי מזה של יישום רגיל. המושגים המקובלים לגבי הזרימה של מידע פנימה והחוצה מהתכנית ולגבי מהות הלולאה הראשית בתכנית אינם תואמים לשיטת עבודה זו. הבה נבחן את המבנה של תכניות התואמות למנהל התצוגה.

תכנית כזו כוללת שני חלקים מרכזיים:

- תהליך עיקרי (main procedure).
- תהליך החלון (window procedure).

לכל התכניות התואמות למנהל התצוגה יש מבנה זהה. כאשר תכנית כזו מתחילה להתבצע, היא עושה את הדברים הבאים:

1. מאתחלת את מנהל התצוגה.
2. קובעת תור הודעות.
3. מפעילה פונקציית חלון (window function) אשר מבצעת הודעות עבור החלון העליון של היישום.
4. יוצרת חלון ברמה הנדרשת.
5. מבצעת לולאה שקוראת הודעות מהתור ומנתבת אותן לפונקציית החלון.

פונקציית חלון יכולה להקרא רק על-ידי מנהל התצוגה. היא מקבלת בפרמטרים שלה הודעה מתור ההודעות ולאחר מכן מבצעת פעולה כלשהי שתלויה בערך של כל הודעה.

הפונקציה ()**main** מקבלת הודעות ומנתבת אותן בדרך המתוארת להלן:

```
Loop until a termination message is received
    Get the next message;
    Dispatch it to the window procedure;
```

בצע לולאה עד אשר מתקבלת הודעת סיום
קרא הודעה מהתור;
שלח את ההודעה אל תהליך החלון;

מנהל התצוגה מתקשר עם התכנית של המשתמש על-ידי העברת הודעה לתור ההודעות. התכנית מוציאה את ההודעה מהתור ושולחת אותה לחלון המתאים על-ידי קריאה לשירות אחר של מנהל התצוגה. תהליך זה ממשיך עד אשר התכנית מסתיימת. אפשר לומר, שברוב הזמן הודעות הינן הדרך היחידה שבה התכנית מקבלת קלט. המבנה של ההודעה שונה לפי הסוג, אך כל ההודעות הן מספרים שלמים (integer).

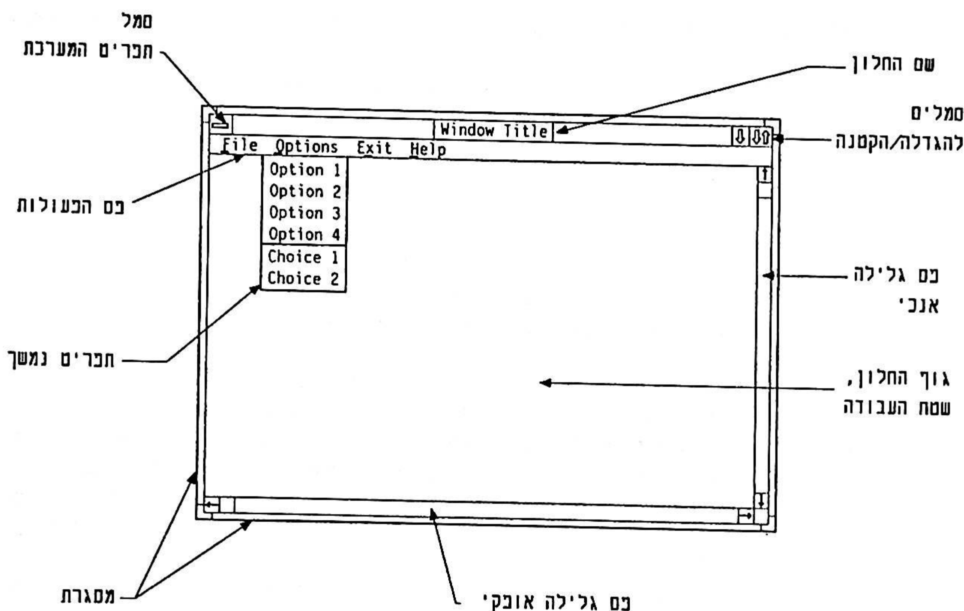
כאשר יישום הפועל תחת מנהל התצוגה מסתיים, הוא חייב לבצע שלוש פעולות:

1. להרוס את החלון.
2. להרוס את תור הודעות.
3. להפסיק את פעולת סביבת החלון המתייחסת ליישום.

מבט מקרוב על החלון

חלון של מנהל התצוגה הוא **תבנית** (frame) בצורת מלבן, אשר כוללת מאפיינים שונים. החלק החשוב בחלון הינו **גוף החלון** (panel body area), או שטח העבודה של המשתמש (client area). בשטח עבודה זה מתבצעות הפעולות הנדרשות על-ידי היישום, כמו הצגת נתונים או ביצוע שגרות. אל החלון אפשר לקשור חלונות בנים כפי שנדרש על-ידי היישום. כל מרכיבי החלון הם בנים של החלון הראשי. הם משמשים לבקרה על פעולות שונות שניתן לבצע בחלון.

מאפיין אחד בתבנית החלון היא **המסגרת** (border), אשר קובעת למעשה את גודלו. המסגרת מאפשרת למשתמש להזיז את החלון או לשנות את גודלו בעזרת העכבר. מאפיין אחר הינו **תפריט המערכת** (system menu). זהו תפריט סטנדרטי, אשר מאפשר למשתמש לבצע את הפעילויות הבאות: לשחזר את החלון לגודלו המקורי, להזיז את החלון, לשנות את גודלו, להקטין את החלון למינימום, להגדיל אותו למקסימום, או לסגור אותו. למרות שהמסגרת מאפשרת להשתמש בשיטה יותר נוחה להזזה או לשינוי של גודל החלון, ניתן לבצע פעילויות אלו גם מתוך תפריט המערכת. כאשר חלון מוקטן למינימום, הוא הופך להיות סמל. כאשר מגדילים חלון לגודל מקסימלי, הוא מכסה את כל פני המסך. סגירה של חלון מוציאה אותו מהמסך, ואם חלון זה הינו **חלון עליון** (top-level window), יתבצע גם המהלך של הפסקת התכנית.



תרשים 3. מערך של חלון סטנדרטי

ברוב החלונות ישנם מאפיינים נוספים: סמלים המאפשרים הקטנה למינימום והגדלה למקסימום של החלון ושם שמזהה את החלון. למרות שניתן לשנות את גודל החלון בעזרת תפריט המערכת, יותר מהר לבצע זאת בעזרת סמלים המיועדים לכך, כי המשתמש יכול להפעיל אותם בעזרת העכבר. השמות מאפשרים את זיהוי החלונות במצבים שבהם יש יותר מאשר חלון אחד במסך. אם התכנית תצטרך זאת, אפשר להוסיף **פס גלילה** (scroll bar) אנכי ואופקי. על-ידי לחיצה (בעכבר) בנקודות המתאימות על גבי פסים אלה ניתן לבצע גלילה של תוכן החלונות כלפי מעלה, מטה, שמאלה וימינה.

מבנה של חלון סטנדרטי במנהל התצוגה נראה בתרשים 3 (זכור, כי אין צורך להשתמש בכל האופציות לגבי כל חלון).

כל תכנית התואמת למנהל התצוגה יוצרת **חלון ראשי** (main window), אשר נמצא ברמה הגבוהה ביותר ובעזרתו המשתמש מזהה את התכנית. סגירה של החלון הראשי מפסיקה את פעולת התכנית.

אפשר לסווג חלונות בשתי קטגוריות כלליות: אבות ובנים. כאשר יישום מתחיל לרוץ, הוא יוצר חלון ראשי אחד או יותר. אם הוא יוצר יותר מחלון ראשי אחד, אז כל החלונות חופפים זה לזה. אולם ישנה אפשרות ליצור חלון אחד בתוך חלון אחר. במקרה זה החלון החדש הוא הבן של החלון הראשי והוא כלול בתוכו. חלון בן יכול בתורו ליצור בן משלו וכן הלאה, עד למגבלה הנכפית בגלל גודל המסך.

כל חלון מסווג לקבוצה כלשהי כללית. חלון המשתייך לקבוצה מסוימת משתמש בתהליכים (procedures) המיועדים לשרת את אותה הקבוצה. ישנם שתי משפחות עיקריות של חלונות: חלונות לשירות הכלל וחלונות פרטיים.

ניתן להשתמש בחלון כללי מכל יישום שהוא. החלונות הסטנדרטיים לשירות הכלל מרכיבים את חלון המשתמש. יש בהם תפריטים, מסגרות, פסי גלילה ועוד. חלון פרטי נוצר על-ידי היישום לשימושו ונמצא בבעלותו בלבד.

כל החלונות מוגדרים ביחס לפינה השמאלית התחתונה (קואורדינטות 0,0). גודלי X ו-Y המקסימליים מוגדרים לכל חלון בצורה דינמית, והם משתנים ביחס לגודל החלון וצורתו, ולפי הרזולוציה של המסך.

יצירת חלון בתכנית

אפשר ליצור חלון בכמה דרכים: כחלון סטנדרטי וכחלון משתמש גמיש, הניתן להתאמה לדרישות מיוחדות. נעמוד תחילה על מספר פעולות מכינות הנדרשות לפני יצירת חלון.

קבלת עוגן

מנהל התצוגה משיג ***מקשר לעוגן*** (anchor block handle) על-ידי קריאה לפונקציה **WinInitialize**. שירותי מנהל התצוגה מחזירים את הערך 0 (NULL) במקרה של כשלון, בניגוד לשירותי ה-API הבסיסיים, אשר מחזירים את הערך 0 במקרה של הצלחה.

המקשר חייב להיות ריק (NULL). הפונקציה מחזירה מחוון (מסוג "רחוק"), אשר מצביע על המקום בזיכרון המשמש את מנהל התצוגה לשם אחסון מידע על סביבת החלון המתייחסת לתכנית היישום. מקום זה בזיכרון הוא העוגן, והמחוון אליו הוא המקשר לעוגן. אם לא ניתן לאתחל את המערכת, מוחזר מקשר ריק. המקשר לעוגן נדרש כפרמטר על- ידי רבים משירותי מנהל התצוגה.

יצירת תור הודעות

לאחר אתחול מערכת החלון, חייבים יישומי מנהל התצוגה ליצור **תור הודעות** (message queue) בעזרת הפונקציה **WinCreateMsgQueue**. פונקציה זו מחזירה מקשר לתור ההודעות, או NULL אם הבקשה נכשלת. מן הראוי להדגיש ש"הודעה" היא מספר שלם ולא מלל, אולם ניתן לפענח מספר זה ולהציג מלל כפי שרצוי.

כל מרכיב של תור ההודעות מוכל בתוך מבנה נתונים (הנקרא QMSG) אשר מכיל את הרכיבים הבאים: מקשר של החלון המקבל, ההודעה, מידע נוסף על ההודעה, הזמן שבו הופקה ההודעה, המיקום של מחוון העכבר.

שיוך חלון לקבוצה

לפני שיוצרים חלון, חייבים לרשום את הקבוצה שאליה הוא שייך. עושים זאת בעזרת הפונקציה **WinRegisterClass**. פונקציה זו מחזירה ערך השונה מאפס במקרה של הצלחה, ואפס במקרה של כשלון.

הפרמטרים של הפונקציה הם: מחוון לעוגן, המחרוזת של שם הקבוצה שיש צורך

לרשום, הכתובת של פונקציית החלון, צורת החלון (style). אפשר לבקש הקצאת שטח בזיכרון מעבר להקצאה הסטנדרטית, או להשאיר את ברירת המחדל.

יצירת חלון סטנדרטי

נעבור עתה ליצירת חלון. נעשה זאת בפקודה create. הדרך הקלה ביותר ליצור חלון היא להשתמש בפונקציה **WinCreateStdWindow**, אשר יוצרת חלון סטנדרטי. היא כוללת פרמטרים אחדים, ביניהם: מקשר לעוגן, מקשר לחלון האב, שיוך לקבוצה, שם החלון, צורת החלון, זיהוי מודול המקור ועוד. הפונקציה מחזירה מקשר לחלון. במקרה של כשלון מוחזר הערך NULL.

כאשר תכנית מתחילה להתבצע, המסך משמש לה כאב, עם ערך מקשר שווה 1. ערך זה מוגדר בעזרת המקרו HWND_DESKTOP. אנו נשתמש בערך זה יותר מאוחר, כאשר נציג דוגמאות שונות.

הערך של style (צורה) קובע מספר מאפיינים של החלון. שמות המקרו הנפוצים רשומים בטבלה בתרשים 4.

לולאת הודעות

כדי שהתכנית תוכל לקבל הודעות היא צריכה להשתמש ב-**WinGetmsg**, אשר יש לה מספר פרמטרים. פונקציה זו מעבירה באופן אוטומטי את ההודעה לפונקציית החלון המתאימה. **WinDispatchMsg** מחזירה את הערך שקיבלה מפונקציית החלון.

ההודעה שנשלפת מהתור מוכנסת למבנה נתונים המוגדר כפרמטר. פרמטר אחר גורם לשלוף הודעות המכוונות לחלון מסוים בלבד. יישום יכול לקבל את כל ההודעות, או תחום הודעות מוגדר. הפונקציה מחזירה אישור ביצוע. כאשר מתקבלת הודעה על הפסקת פעולה, היא מחזירה הודעת אי-ביצוע.

שם המקרו	משמעות
WS_VISIBLE	קבע חלון שניתן לראות אותו
WS_MINIMIZED	הקטן את החלון למינימום
WS_MAXIMIZED	הגדל את החלון למקסימום
FS_TITLEBAR	כלול פס כותרת
FS_SYSMENU	כלול תפריט מערכת
FS_VERTSCROLL	כלול פס גלילה אנכי
FS_HORZSCROLL	כלול פס גלילה אופקי
FS_SIZEBORDER	כלול גבולות השוליים
FS_BORDER	השתמש בשוליים דקים
FS_MINBUTTON	כלול סמל המקטין למינימום
FS_MAXBUTTON	כלול סמל המגדיל למקסימום
FS_MINMAX	כלול סמל מקטין וסמל מגדיל

תרשים 4. המקרו הנפוצים ביותר לפרמטר style בפונקציה WinCreateStdWindow

סיום הפעולה של התכנית

לפני שהתכנית מסתיימת, היא צריכה לבצע שלושה דברים: לסגור את כל החלונות הפעילים, לסגור את תור ההודעות ולהפסיק את הפעולה של מישק מערכת החלון שנוצר. פעולות אלו נעשות על-ידי הפונקציות **WinDestroyWindow**, **WinDestroyMsgQueue** ו-**WinTerminate**.

בפונקציה הראשונה, אחד מהפרמטרים הוא המקשר לחלון שיש לסגור אותו. בשניה - פרמטר מציין את המקשר לתור ההודעות שיש להרוס. ולבסוף, מערכת החלון מתנתקת על-ידי קריאה ל-**WinTerminate** עם המקשר לעוגן, המציין את כתובת החלון בזיכרון.

פונקציית החלון

כל התכניות התואמות למנהל התצוגה חייבות להעביר לו את הכתובת של פונקציית החלון שתקבל את ההודעות, והיא אשר מעבירה אותן לתכנית. המתכנת מכריז, נותן לפונקציה את השם **window_pgm** למשל, ומגדיר אותה עם מספר פרמטרים: מקשר לחלון שצריך לקבל את ההודעה ומספר שלם המכיל את ההודעה עצמה. אפשר להעביר מידע נוסף באמצעות אחד או שני פרמטרים אפשריים.

מנהל התצוגה יכול להפיק סוגים שונים של הודעות. ההודעות השכיחות יותר מופיעות בטבלה שבתרשים 5. פונקציית החלון אינה צריכה לטפל בכל ההודעות שהיא מקבלת. עליה לעסוק רק באלה אשר חשובים לתפעול תקין של היישום. שאר ההודעות שפונקציית החלון אינה מטפלת בהן נשלחות חזרה למנהל התצוגה בעזרת **WinDefWindowProc** לשם ביצוע ברירות המחדל של המערכת. למעשה, הפונקציה מחזירה למנהל התצוגה את הפרמטרים שהועברו אליה קודם לכן.

שם המקרו	משמעות
WM_BUTTON1DOWN	לחצן 1 לחוץ
WM_BUTTON1UP	לחצן 1 שוחרר
WM_BUTTON1DBLCLK	לחיצה כפולה על לחצן 1 ...לחצנים 2, 3...
WM_CHAR	נלחץ מקש במקלדת
WM_CREATE	נוצר חלון
WM_DESTROY	מתבצעת הריסה של החלון
WM_ERASEBACKGROUND	אישור למחיקת בקשת רקע
WM_HSCROLL	גלילה אופקית
WM_MOVE	מתבצעת הזזה של החלון
WM_MOUSEMOVE	היתה תנועה בעכבר
WM_PAINT	חידוש תצוגת החלון
WM_SHOW	הצגה של החלון, או הסרה מהמסך
WM_SIZE	מתבצע שינוי גודל בחלון
WM_VSCROLL	גלילה אנכית
WM_QUIT	החלון מופסק

תרשים 5. ההודעות השכיחות

שלא כמו תכניות OS/2 רגילות, תכניות התואמות למנהל התצוגה חייבות לכלול קובץ להגדרת המודול (MDF) בשורת הפקודה המפעילה את תכנית הקישור. הסיבה העיקרית לכך היא, שיישום התואם למנהל התצוגה דורש **שטח מחסנית** (stacksize) גדול יותר מזה הניתן לו על-ידי ברירת המחדל של המערכת. הדוגמה בפרק זה מקצה 4096 בתים, אך סביר להניח שתכניות אמיתיות יצטרכו מעבר לכך. כדאי גם לציין את מספר הבתים שהתכנית צורכת לשימוש עצמי (heapsize). תכנית הדוגמה מקצה 1024 למטרה זו, אבל יש להקצות בהתאם לגודל התכניות. צריך לכלול את המשפט EXPORTS שמציין את השם של פונקציית החלון. הקובץ להגדרת המודול של שלד תכנית הדוגמה נראה כך:

```
NAME        hello1 WINDOWAPI
HEAPSIZE    1024
STACKSIZE   4096
EXPORTS     MyWindowProc
```

בשלב ההידור של תכנית תצוגה, יש לציין למהדר אופציות השונות מאלו של תכנית סטנדרטית. יש לקרוא בעיון את ההוראות בספר ההדרכה של המהדר שבו כותבים את התכנית (שפת C או אסמבלר).

תכנית דוגמה לתכנות חלונות

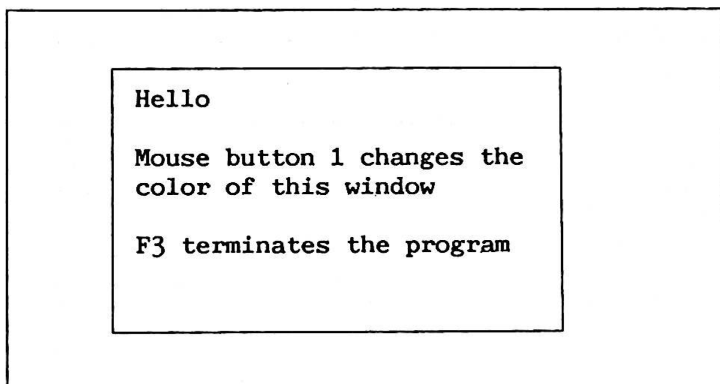
ראינו עד עתה את השירותים הדרושים לאתחול ולהרצה של יישום פשוט המבוסס על חלונות, ועתה נראה תכנית שלמה, אשר נכנה אותה HELLO1.

התכנית HELLO1 מדגימה כמה מהעקרונות והמרכיבים של תכנית יישום בבקרת מנהל התצוגה (PM application):

1. תכנית מקור, אשר המבנה הכללי שלה זהה בכל יישומי מנהל התצוגה (ואשר תואמת את מישק התצוגה המוגדר על-ידי SAA).
2. תקשורת עם OS/2, עם יישומי תצוגה אחרים ועם היישום עצמו.
3. היענות וידידותיות למשתמש.

למעשה, לפנינו שלד של תכנית, אשר ניתן להפעילה כפי שהיא, לשם הדגמה ותרגול. אפשר גם להכניס בה שינויים ולהתאים אותה לדרישות של יישום ספציפי כלשהו. התכנית כתובה בשפת C/2. ההערות בשולי כל הוראה ניתנו לשם הסבר. כפי שניתן לראות, חלק גדול מהתכנית מוקדש לקריאה והפעלה של שגרות סטנדרטיות של המערכת.

התכנית יוצרת חלון על פני המסך ומציגה בו מלל בתווים לבנים על רקע אדום. ראה תרשים 6. כאשר המשתמש מציב את המחוון בחלון ומקיש על לחצן 1 בעכבר (בד"כ זהו הלחצן השמאלי), צבע הרקע בחלון מוחלף בין אדום לכחול. אם יש חלון שמוצג במסך וחלון אחר שמוצג עליו, אז כאשר מסירים את החלון העליון, החלון התחתון יופיע מחדש. המקש היחיד במקלדת שנשתמש בו הוא F3, כדי להפסיק את התכנית.



תרשים 6. תמונת החלון ביישום HELLO1

תכנית הדוגמה מוצגת ב-2 עמודים בתרשימים 7(1), 7(2). בראש התכנית אנו מוצאים את ההוראה include#, אשר מציינת שיש להשתמש בקובץ OS2.h כקובץ כותרת. הוא כולל הגדרות של מבני נתונים וקריאות לשגרות API רבות, ולכן אנו מבקשים לשלב בתכנית רק את הקטעים הנחוצים. זאת אנו עושים בהוראה define INCL_WININPUT#. ההוראה define ID_WINDOW 255# מציבה את הערך 255 במשתנה. בתכנית מעשית יש הוראות רבות מסוגים אלה ומקובל להכילן בקובץ סטנדרטי, אשר ישמש אחר-כך יישומים רבים.

הפונקציה MyWindowProc הינה פונקציית החלון. היא מטפלת בקלט, בפלט ובתצוגה, ומהווה למעשה את הקשר עם המשתמש. כאן מוצגת התבנית שלה.

הפונקציה המרכזית של התכנית (בשפת C) הינה main. זוהי נקודת הכניסה אל תכנית יישום הפועלת בבקרת מנהל התצוגה. כל תכנית הפועלת בחלון חייבת לכלול אותה. שתי ההוראות שלפניה ו-5 ההוראות שאחריה הן הגדרות של משתנים, אשר תואמים להגדרות שבקובץ המוכלל (include).

הפונקציה **main()** מבצעת את הדברים הבאים:

- יוזמת את הקשר בין מנהל התצוגה לבין התכנית.
- רושמת את קבוצת החלונות החדשה.
- יוצרת חלון.
- מבצעת את לולאת ההודעות.

הפונקציה WinInitialize מכינה את סביבת העבודה ומשאירה מקשר לעוגן המתייחס ל-thread המופעל. מקשר זה דרוש כפרמטר גם לפונקציות אחרות.

השלב הבא הוא יצירה של תור הודעות על-ידי הפונקציה WinCreateMsgQueue. הפונקציה מחזירה את המקשר לתור. לכל thread חייב להיות תור הודעות נפרד, שבו הוא יוכל לקבל קלט חיצוני.

הפונקציה WinRegisterClass רושמת את סיווג החלון. ברירת המחדל היא "פרטי". הפונקציה מעבירה ל-OS/2 את שם קבוצת החלון MyWindow ואת שם פונקציית החלון של המשתמש MyWindowProc.

```
#define INCL_WININPUT
#include <os2.h>                               /* Select part of header        */
                                               /* PM header file               */
#define ID_WINDOW    255

/***********************************************************************/
/* Function prototypes                                                 */
/***********************************************************************/
MRESULT EXPENTRY MyWindowProc( HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2 );
void cdecl main( void );


HAB   hab;                                     /* Define parameters by type    */
COLOR colorWnd = CLR_RED;                      /* PM anchor block handle       */
                                               /* Window Color                 */
/********************* Start of main procedure ***********************/
void cdecl main(  )
{
  HMQ  hmq;
  HWND hwndClient;                             /* Message queue handle         */
  HWND hwndFrame;                              /* Client area window handle    */
  QMSG qmsg;                                   /* Frame window handle          */
  ULONG flCreate;                              /* Message from message queue   */
                                               /* Window creation control flags*/
  hab = WinInitialize( NULL );                 /* Initialize PM                */
  hmq = WinCreateMsgQueue( hab, 0 );           /* Create a message queue       */

  WinRegisterClass(                            /* Register window class        */
     hab,                                      /* Anchor block handle          */
     "MyWindow",                               /* Window class name            */
     MyWindowProc,                             /* Address of window procedure  */
     0L,                                       /* No special Class Style       */
     0                                         /* No extra window words        */
     );

  flCreate = FCF_TASKLIST;                     /* Set Frame Control Flag       */

  hwndFrame = WinCreateStdWindow(
               HWND_DESKTOP,                   /* Desktop window is parent     */
               0L,                             /* No Class Style               */
               (PULONG)&flCreate,              /* Frame control flag           */
               "MyWindow",                     /* Client window class name     */
               "",                             /* No window text               */
               0L,                             /* No special class style       */
               NULL,                           /* Resource is in .EXE file     */
               ID_WINDOW,                      /* Frame window identifier      */
               (PHWND)&hwndClient              /* Client window handle         */
               );

  WinSetWindowPos( hwndFrame,                  /* Set the size and position of */
                   HWND_TOP,                   /* the window before showing.   */
                   100, 100, 250, 200,
                   SWP_SIZE | SWP_MOVE | SWP_ACTIVATE | SWP_SHOW
                 );

/***********************************************************************/
/* Get and dispatch messages from the application message queue        */
/* until WinGetMsg returns FALSE, indicating a WM_QUIT message.        */
/***********************************************************************/
  while( WinGetMsg( hab, (PQMSG)&qmsg, (HWND)NULL, 0, 0 ) )
    WinDispatchMsg( hab, (PQMSG)&qmsg );

  WinDestroyWindow( hwndFrame );               /* Tidy up...                   */
  WinDestroyMsgQueue( hmq );                   /* and                          */
  WinTerminate( hab );                         /* terminate the application    */
}
/********************** End of main procedure ************************/
```

```
/********************* Start of window procedure *********************/
MRESULT EXPENTRY MyWindowProc( HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2 )
{
  HPS    hps;                     /* Presentation Space handle    */
  RECTL  rc;                      /* Rectangle coordinates        */
  POINTL pt;                      /* String screen coordinates    */

  switch( msg )
  {
    case WM_PAINT:
      /*************************************************************/
      /* Window contents are drawn here in WM_PAINT processing.    */
      /*************************************************************/
                                  /* Obtain a cached micro PS     */
      hps = WinBeginPaint( hwnd, NULL, &rc );
      WinFillRect( hps, &rc, colorWnd );/* Fill invalid rectangle     */
      pt.x = 10L; pt.y = 190L;        /* Set the text coordinates,    */
      GpiSetColor( hps, CLR_WHITE );     /* Set color of the text        */
      GpiCharStringAt( hps, &pt, 5L, (PSZ)"Hello" ); /* Draw Text       */
      pt.y = 170L;
      GpiCharStringAt( hps, &pt, 26L, (PSZ)"Mouse button 1 changes the" );
      pt.y = 160L;
      GpiCharStringAt( hps, &pt, 20L, (PSZ)"color of this window" );
      pt.y = 140L;
      GpiCharStringAt( hps, &pt, 25L, (PSZ)"F3 terminates the program" );
      WinEndPaint( hps );                      /* Drawing is complete   */
      break;
    case WM_BUTTON1DOWN:
      /*************************************************************/
      /* Mouse button 1 has been clicked, so first ensure we have the */
      /* input focus and then toggle the window color between RED and */
      /* BLUE and cause the window to be redrawn.                  */
      /*************************************************************/
      WinSetFocus( HWND_DESKTOP, hwnd );
      colorWnd = ( colorWnd == CLR_RED ) ? ( CLR_BLUE ) : (CLR_RED );
      WinInvalidateRect( hwnd, NULL, TRUE );
      break;
    case WM_CHAR:
      /*************************************************************/
      /* Character input is processed here.                        */
      /* The first two bytes of message parameter 2 contain        */
      /* the character code.                                       */
      /*************************************************************/
      if( SHORT2FROMMP( mp2 ) == VK_F3 )    /* If the key pressed is F3,*/
        WinPostMsg( hwnd, WM_QUIT, 0L, 0L );/* post a quit message to   */
      break;                             /* end the application.     */
    case WM_CLOSE:

      /*************************************************************/
      /* This is the place to put your termination routines        */
      /*************************************************************/
      WinPostMsg( hwnd, WM_QUIT, 0L, 0L );  /* Cause termination       */
      break;
    default:
      /*************************************************************/
      /* Everything else comes here.  This call MUST exist         */
      /* in your window procedure.                                 */
      /*************************************************************/

      return WinDefWindowProc( hwnd, msg, mp1, mp2 );
  }
  return FALSE;
}
/********************** End of window procedure **********************/
```

בשלב הבא יש ליצור חלון סטנדרטי על-ידי הפונקציה WinCreateStdWindow, אשר שם הקבוצה שלו הוא MyWindow. אנו מציינים את ID_WINDOW, אשר משמש כמזהה לחלון (מספר 255). בדוגמה זו אין מכניסים טכסט עבור שם החלון. את החלון מציבים על פני המסך על-ידי הפונקציה WinSetWindowPos. אנו מציינים שהחלון הזה יהיה העליון (HWND_TOP). הפרמטר הבא כולל שני זוגות של מספרים (pels) המציינים את הקואורדינטות של נקודת ההתחלה ואת הגודל של החלון. פרמטרים אחרים מציינים שהחלון יופעל וגם ייראה.

הלולאה הפעילה של התכנית המרכזית היא קבלת הודעות (WinGetMsg) והעברתן (WinDispatchMsg) לפונקציית החלון MyWindowProc, אשר מוגדרת ב-qmsg. בדוגמה זו יש חלון אחד בלבד, אך בתכניות מורכבות יותר יכול להיות יותר מחלון אחד, ובהתאם - יותר מפונקציית חלון אחת. לולאת ההודעות מפסיקה לפעול כאשר מתקבלת הודעה WM_QUIT מהתכנית, או כאשר המשתמש מקיש F3.

את התכנית מסיימים בשלבים, על-ידי שלוש פונקציות: הריסת המקשר לחלון, הריסת המקשר לתור ההודעות ולבסוף - סיום התכנית.

הפונקציה החשובה ביותר הינה פונקציית החלון. היא מקבלת הודעות שנשלחו על-ידי מנהל התצוגה ומבצעת לפיהן את הפעולות המתאימות. ניתן לראות במשפט switch את ההודעות שמנהל התצוגה יטפל בהן.

בזמן שהחלון נוצר, אפשר להפיק את ההודעה WM_CREATE לפונקציית החלון. דבר זה מאפשר לתכנית לאתחל ערכים, או לבצע פעולות ראשוניות. מנהל התצוגה מאפשר למשתמש להזיז את החלון ולשנות את גודלו, או לכסות חלק ממנו על-ידי חלון אחר. המשמעות של הפעולה האחרונה היא, שבשלב מסוים יהיה צורך לצייר מחדש את החלון המקורי, או את חלקו. בתכנית זו לא הגדרנו פעולה באפשרות זו. מנהל התצוגה מפיק את ההודעה WM_PAINT בכל פעם שיש צורך לחדש את תוכן החלון. הפעולות שיש לעשות במקרה זה מתוארות בהרחבה ומדגימות כיצד נכתב הטכסט בחלון.

ההודעה WM_BUTTON1DOWN גורמת לפעולה הנובעת מלחיצה על כפתור 1 בעכבר. ההודעה WM_CHAR מגיבה ללחיצה של המשתמש על F3, כדי להורות על הפסקת התכנית. הודעה זו מופקת בכל הקשה שהיא במקלדת.

כאשר המשתמש מסיים את התכנית מתפריט המערכת במסך, הוא גורם לכך ש-OS/2 תשגר הודעה WM_CLOSE אל היישום, אשר מתרגם אותה להודעת WM_QUIT, אשר תגרום להפסקת התכנית.

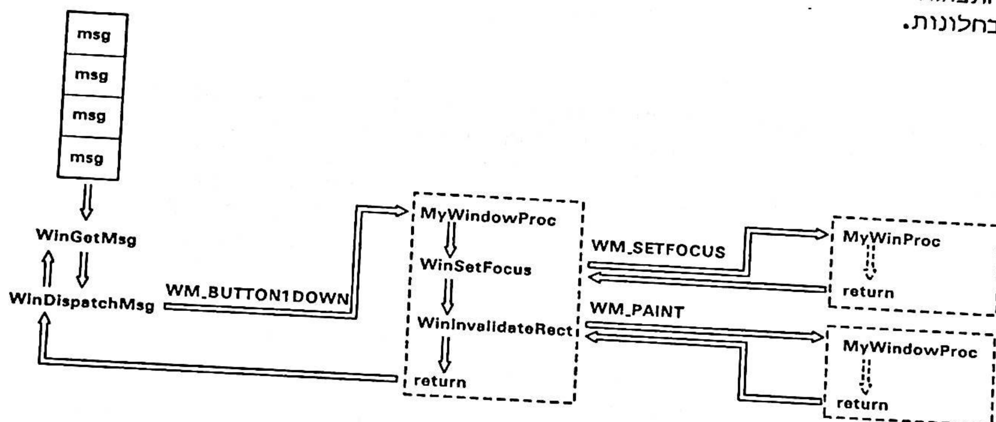
כל הודעה שהתכנית אינה רוצה לעבד, כמו הוראת גלגול למשל, מוחזרת למנהל התצוגה דרך השירות WinDefWindowProc. הודעות אלו תטופלנה בהתאם לברירות המחדל של המערכת.

מנהל התצוגה מול שירותי גרעין

במהלך הסבר זה ניתן לראות שיש חפיפה מסוימת בין שירותי מנהל התצוגה לבין **שירותי הגרעין** (core/kernel services) של OS/2. כאשר כותבים בתכנית התואמת למנהל התצוגה יש להמנע משימוש בשירותי ה-**Vio**, ה-**Kbd** וה-**Mou**. במקום זאת רצוי להשתמש בשירותים המתאימים של מנהל התצוגה. עם

זאת, מותר להשתמש בשירותי הגרעין של Dos, ובמיוחד באלה התומכים בתקשורת בין תהליכים ובאלה התומכים בבקרה על התקנים.

אם תכנית משתמשת בשירותי Vio, Kbd ו-Mou, היא תרוץ בקבוצת מסך משלה, ולא בחלון של מנהל התצוגה. השימוש העיקרי בשירותים אלה בסביבת מנהל התצוגה הוא בתכניות שירות, ובמיוחד באלו שאינן דורשות תמיכה של עבודה בחלונות.



תרשים 7. התהליך הרקורסיבי של ביצוע החלונות בתכנית HELLO1

```
case WM_VSCROLL:
     .
     .
     .
     .
   switch( HIUSHORT( mp2))
     {
      case SB_LINEDOWN:
           Scroll( 0, WNDROW );
           break;

      case SB_LINEUP:
           Scroll( 0, -WNDROW );
           break;

      case SB_PAGEDOWN:
           Scroll( 0, WndDepth-2 );
           break;

      case SB_PAGEUP:
           Scroll( 0, -( WndDepth-2 ));
           break;

      case SB_SLIDERPOSITION:
           Scroll( 0,  LOUSHORT( mp2 ) - orig.y );
           break;

      default:
           break;
     }
}
```

תרשים 8. תהליך גילגול של חלון במסך

נספח ב

מנהל בסיס הנתונים

העלאת העוצמה של מערכות מחשב אישיות, גרמה לדרישה של המשתמשים למערכות ניהול של בסיסי נתונים. כמות הנתונים שניתן לאחסן ולנהל במחשבים אישיים גדלה בצורה מהירה והמורכבות של הנתונים ומגוון היישומים המבוססים עליהם גדלים בצעדי ענק. משתמשים שפעם היו מרוצים מתכניות פשוטות לניהול קבצים חיפשו מערכת חזקה לניהול בסיס נתונים, שבאמצעותה יוכלו לנהל ולשלוט בנתונים שלהם. מפתחי יישומים דורשים גם הם את המאפיינים של מערכות לניהול בסיס נתונים, אשר פועלות במערכות גדולות.

מנהל בסיס הנתונים (Database Manager), הנכלל בג'ירסה המורחבת של OS/2, הינו חבילת תוכנה אשר נועדה הן עבור משתמשים והן עבור מפתחי יישומים. מנהל בסיס הנתונים הטבלאי/יחסי (Relational Database Manager) מכיל את הפונקציות הדרושות לניהול בסיסי נתונים ותומך בשפת השאילתות המובנית SQL של יבמ. הטכנולוגיה והארכיטקטורה של מנהל בסיס הנתונים מספקות את הדברים הבאים:

- רמת ביצועים גבוהה.
- גישה למשתמש אחד וגישה בו-זמנית למספר משתמשים.
- שלמות של הנתונים.
- אמצעים להגנת הנתונים.

מנהל בסיס הנתונים מבוסס על שפת השאילתות SQL ועל מודל בסיס הנתונים הטבלאי שפותח על-ידי E.F. Codd במרכז המחקר של יבמ בסאן-חוזה, ארה"ב. היתרון העיקרי שלו הוא בהפרדה הברורה בין תפיסת המשתמש לגבי הנתונים, לבין הצורה שהם מאוחסנים באמת. בסביבת המחשב האישי, ידידותיות של היישום הינה הדגש העיקרי. ואמנם, המודל הטבלאי של הנתונים פשוט וקל להבנה ולשימוש גם על-ידי מי שאינו מתמחה במחשבים: המשתמש עוסק בהיבט של השימוש בנתונים ומנהל בסיס הנתונים עוסק בהיבט של האחסנה, העדכון והאחזור של הנתונים. התוכנה מפעילה מנגנוני גישה יעילים כדי להגיע לביצוע אופטימלי של איחזור המידע.

שפת SQL, שפותחה כעבודת מחקר בנושא בסיסי נתונים, הפכה לסטנדרט בתעשייה. השפה קלה ללמידה, אך בעלת עוצמה רבה בביטוי שאילתות מורכבות. משפט יחיד ב-SQL יכול לבצע את אותן פעולות, שיכולות לבצע שורות רבות של פקודות בשפת תכנות מקובלת.

שפה זו נתמכת על-ידי מוצרי תוכנה של יבמ למחשבים בארכיטקטורת 370 ומחשבי AS/400:

IBM Database 2 (DB2)
IBM Structured Query Language/Data System (SQL/DS)
IBM SQL/400

מנהל בסיס הנתונים ב-OS/2 תומך במישק בסיס הנתונים SQL של SAA - ארכיטקטורת היישומים הבין-מערכתית של יבמ. SAA מספקת מישק סטנדרטי לשלוש סביבות המיחשב העיקריות של יבמ: מחשבים בארכיטקטורת 370, מערכות בינוניות ומחשבים אישיים.

הכלי ההידברותי למשתמש הקצה הוא **מנהל השאילתות** (Query Manager). כלי זה מאפשר הכנסת נתונים, עריכה, ביצוע שאילתות, יצירת דו"חות ותמיכה בהתאמת יישומים לצורכי הלקוח (customization). מנהל השאילתות ב-OS/2 תומך במישק השאילתות של SAA ותואם למערכות התומכות בארכיטקטורה זו.

פרק זה מתאר את הפונקציות של מנהל בסיס הנתונים ב-OS/2.

המודל הטבלאי של יבמ

לאחר שפותחה התיאוריה הבסיסית על המודל הטבלאי (יחסי - relational), יבמ פיתחה מספר אבות טיפוס המבוססים על המודל. אב הטיפוס הידוע ביותר נקרא System R, אשר פותח במעבדת המחקר בסאן-חוזה ושפת SQL פותחה כחלק מעבודה זו. פותחו גם טכניקות אופטימיזציה והידור של שאילתות, בקרת גישה במקביל (concurrency control), ופרוטוקולים לנעילה ורישום. אב הטיפוס System R סיפק את הבסיס למחקר ולפיתוח של מערכות לניהול בסיס נתונים מבוזר (Distributed DataBase Management System - DDBMS).

המוצרים SQL/DS ו-DB2 פועלים מאז ראשית שנות ה-80. מנהל בסיס הנתונים ב-OS/2 EE נהנה מכל הטכנולוגיות, ממאמצי המחקר, ומכל הניסיון שנצבר במוצרים שכבר קיימים. הוא תוכנן במיוחד לרוץ בסביבת OS/2 ומנצל טכנולוגיות חדישות, כדי להשיג את הביצועים הטובים ביותר האפשריים. היתרונות של מנהל בסיס הנתונים ב-OS/2 EE בולטים בנושאים הבאים:

ביצועים: רמת ביצועים גבוהה היתה יעד תכנון עיקרי של מנהל בסיס הנתונים. כדי להשיג יעד זה השתמשו בטכנולוגיות חדשות לאופטימיזציה של ביצועים, אלגוריתמים משותפים (join algorithms), וטכניקות מתאימות לניהול מאגרים. השיטות למיעון לפי אינדקס ולנעילה סיפקו רמה גבוהה של גישה במקביל תוך שמירה על רמה גבוהה של ביצועים.

ארכיטקטורת יישומים בין מערכתית (SAA): היתרונות של SAA הם חלק ממנהל בסיס הנתונים של OS/2 EE. התחביר, הסמנטיקה והארכיטקטורה של פונקציות מישק בסיס הנתונים תוכננו כדי שיהיו זהים לאלה הנמצאים במוצרים אחרים.

שמירה על שלמות הנתונים: השתמשו בטכניקות חדשות לניהול תנועות ולרישום (logging) כדי להבטיח את שלמות הנתונים (data integrity) בעת תקלה.

מערכות לניהול בסיס נתונים מבוזר (DDBMS): רוב הפונקציות אינן כלולות בגירסה הראשונה, אך המוצר עצמו תוכנן בצורה כזו שניתן לשלב בו פונקציות אלו בשלב מאוחר יותר. ההתאמה בארכיטקטורות של מערכות בסיסי הנתונים הטבלאיים של יבמ, מאפשרת את הביזור של בסיס הנתונים בין מוצרי תוכנה הפועלים במחשבים שונים.

מבנה המערכת

המשתמש יכול להפעיל את הפונקציות של מנהל בסיס הנתונים בשלוש דרכים:

- ניתן להשתמש במישק ההידברותי הנקרא מנהל השאילתות, כדי לבצע באופן מזדמן שאילתות, הכנסת נתונים, עדכון, הגדרת נתונים ויישומי דו"חות.
- ניתן להשתמש בפונקציות של מנהל השאילתות כדי ליצור יישומים עם טפסים, תפריטים, שאילתות, דו"חות ופרוצדורות, אשר מותאמים למשתמש הסופי. המשתמש יכול לבצע יישומים אלה שוב ושוב, על-ידי הזנת פרמטרים מתאימים.
- ניתן להשתמש בשפות תכנות מקובלות כמו C, כדי לקבל שירותים ישירים מפונקציות שונות של מנהל בסיס הנתונים. קדם מהדרים (precompilers) ותכניות קשירה (binders) משמשים ליצירת קשר בין תכניות המשתמש לבין **שירותי בסיס הנתונים**.

כל הגישות לבסיס הנתונים עצמו מבוצעות על-ידי מרכיב של מנהל בסיס הנתונים הנקרא **שירותי בסיס הנתונים** (Database Services). זהו המנוע של מנהל בסיס הנתונים, אשר מטפל בנתונים המאוחסנים בבסיס הנתונים ובפונקציות של SQL המגיעות דרך מישק בסיס הנתונים. הוא מפיק תכניות לגישה אופטימלית ותומך בפונקציות המאפשרות הזנת תנועות, גישה בו-זמנית, אחסון של נתונים וניהול של מאגרים. באמצעות שגרות API הוא גם מטפל בשירותים כמו יבוא, יצוא, גיבוי, אישוש, ארגון מחדש ורישום סטטיסטיקה של הרצות.

אם בסיס הנתונים הינו מרוחק ונמצא ברשת תקשורת מקומית (LAN), יקראו שירותי בסיס הנתונים למרכיב **שירותים מרוחקים** (Remote Services), שאינו זמין עדיין, כדי שיטפל בגישה מרחוק. מרכיב השירותים המרוחקים משתמש בפרוטוקול "תקשורת מתקדמת בין תכניות" (APPC) של רשת ה-SNA כדי ליצור קשר עם בסיס נתונים מרוחק. APPC נתמך על-ידי מנהל התקשורת של OS/2 EE, שהינו מרכיב נוסף הנכלל במהדורה המורחבת של OS/2.

בהמשך נתאר פונקציות עיקריות המסופקות עם מישק בסיס הנתונים.

מישק בסיס הנתונים

טבלאות נתונים

שפת SQL מאפשרת להגדיר בדרך פשוטה את הטבלאות המרכיבות את בסיס הנתונים. פשטות זו נובעת מהעובדה שמבנה הנתונים אינו מכיל מצביעים, אינדקסים, או מסלולי גישה כלשהם מוגדרים.

בסיס הנתונים במודל הטבלאי מורכב מטבלאות, אשר יש להגדיר בעת בניית בסיס הנתונים או לאחר מכן, כאשר רוצים להוסיף טבלה חדשה. הגדרת טבלה כוללת את שמה, את שמות השדות (עמודות) המרכיבות אותה ואת התכונות של השדות. בתרשים מוצג מבנה של טבלאות שעל פיהן נוכל לרשום את הסכימה הטבלאית של בסיס הנתונים. לשם דוגמה נניח שקיימות במערכת שתי טבלאות

המתייחסות למחברים ולספרים שהם כתבו. נערוך רשימות לפי חתכים ומיונים שונים ונערוך סיכומים שונים על-פי הנתונים הכלולים בהם.

AUTHORS(AUTHORID, LASTNAME, FIRSTNAME, ROYALTIES)

AUTHORID זיהוי המחבר	LASTNAME שם משפחה	FIRSTNAME שם פרטי	ROYALTIES תמלוגים
*			

BOOKS(TITLE, COPIES, TYPE, AUTHORID)

TITLE כותר (שם הספר)	COPIES מס' עותקים שנמכרו	TYPE סוג	AUTHORID זיהוי המחבר
			*

שפה לטיפול בנתונים

ב-SQL קיימות 4 פונקציות עיקריות לצורך הטיפול בנתונים: SELECT, UPDATE, INSERT, SELETE.

הפונקציה SELECT היא החשובה והשימושית בכולן. היא משמשת לאיחזור נתונים ולשם הצגת VIEW. המבנה הכללי של ההוראה SELECT:

```
SELECT       attribute list      שמות שדות לשליפה (או * עבור כל השדות)
FROM         table names         שם טבלה, או טבלאות, שמתוכן יש לשלוף
[WHERE       qualification]      תנאי שליפה
[ORDER BY    attribute-name]     סדר מיון
                                 [ ]   מסמן משפט אופציונלי
```

המשתמש במערכת מעוניין לעתים רק בחלק מהנתונים הכלולים בטבלאות השונות ולרוב הוא משתמש במספר טבלאות לשם הצגת השאילתות. בדרך המקובלת היה עליו להגדיר מדי פעם מחדש את אוסף הנתונים (טבלאות, שדות ותנאי שליפה) או ליצור לעצמו טבלה ממשית נפרדת. כדי להקל עליו, מאפשרת המערכת להציג לו טבלה לוגית, שהיא **תצפית** קבועה (VIEW) של הנתונים שנדרשים על ידו. **טבלה לוגית** אינה קיימת באופן ממשי בבסיס הנתונים, אלא מוגדרת על פי הטבלאות הממשיות, והמשתמש פועל בה כאילו היתה טבלה ממשית. שיטה זו

מגדילה את אי התלות בנתונים ואת יעילות העבודה. השיטה תורמת גם לבטיחות הנתונים בכך שהיא מאפשרת להציג למשתמש רק את הנתונים המותרים לו ומסתירה מפניו את השאר.

ההוראה SELECT מאופיינת בכך שהיא שולפת מספר שורות (רשומות) בבת-אחת ומפיקה כפלט רשימה או דו"ח שלם. בשפות אחרות יעד השליפה הוא רשומה בודדת, והפקת דו"ח נוצרת על-ידי איחזור בתהליך חוזר (לולאה).

נציג מספר דוגמאות:

א. עריכת דו"ח מטבלת המחברים, הכולל את שם המשפחה והשם הפרטי של המחבר. המיון לפי שם משפחה ושם פרטי בסדר עולה (ASCending):

```
SELECT      LASTNAME,FIRSTNAME
FROM        AUTHORS
ORDER BY    LASTNAME ASC, FIRSTNAME ASC
```

כאשר נרצה לקבל רשימה לא ממוינת של המחברים ששמם הפרטי הוא 'SMITH', נכתוב:

```
SELECT   LASTNAME, FIRSTNAME
FROM     AUTHORS
WHERE    FIRSTNAME = 'SMITH'
```

ב. נשתמש באופטור קבוצתי הפועל על קבוצת הרשומות שנבחרה מתוך טבלה אחת או יותר. נוכל להשתמש באופרטורים שונים, כמו:

```
COUNT, SUM, AVG, MAX, MIN
```

לדוגמה, נחשב עבור כל המחברים את ממוצע התמלוגים ונכפול ב-1/2.

```
SELECT   AVG(ROYALTIES) * 0.5
FROM     AUTHORS
```

ג. אפשר לקבל סיכומי ביניים לפי קריטריון של סוג, למשל:

```
SELECT      TYPE, SUM(COPIES)
FROM        BOOKS
GROUP BY    TYPE
            HAVING TYPE='SF' OR TYPE='FI'
```

ד. אפשר לרכז נתונים ממספר טבלאות, עד 15. הקשר בין הרשומות ייעשה באמצעות שדה (עמודה) מזהה הנמצא בשתי הטבלאות, אשר במקרה שלפנינו הוא AUTHORID.

```
SELECT   LASTNAME, TITLE
FROM     AUTHORS A, BOOKS B
WHERE    A.AUTHORID = B.AUTHORID
   AND   TYPE = 'SF'
   AND   FIRSTNAME = 'JOHN'
```

ה. במערכת SQL אפשר לבנות שאילתות מורכבות מאוד תוך שימוש **בתת-שאילתות** (Subquery). הפלט של תת-השאילתה, שהינו למעשה טבלה, נמסר כקלט לשאילתה הראשית. נציג את השימוש בטכניקה זו.

```
SELECT   FIRSTNAME, LASTNAME
FROM     AUTHORS
   WHERE ROYALTIES >
      (SELECT AVG (ROYALTIES)
      FROM AUTHORS
      AND
      AUTHORID = ANY (SELECT AUTHORID
                      FROM BOOK
                      WHERE TYPE = 'SF'))
```

משפטי UPDATE, INSERT ו-DELETE יכולים להכיל את הביטוי WHERE. לדוגמה, משפט UPDATE: כל המחברים שמחברים ספרים מסוג SF, יקבלו תוספת תמלוגים של 10%.

```
UPDATE   AUTHORS
SET      ROYALTIES = ROYALTIES * 1.1
WHERE    AUTHORID = ANY (SELECT AUTHORID
                         FROM BOOKS
                         WHERE TYPE = 'SF')
```

מישק התכנות

ההוראה SELECT מפיקה לעתים יותר משורה אחת. כדי להקצות את הזיכרון הדרוש לאחסנת הפלט הזה, יש לרשום את ההוראה DECLARE CURSOR לפני הוראת SELECT. הוראת FETCH מאפשרת לטפל בשורה אחת מדי פעם. היא מעבירה את הנתונים אל התכנית בהתאם לשמות המשתנים המוגדרים בה.

לדוגמה, משפטי SQL בתכנית בשפת C, המעבירים פלט למשתנים:

```
EXEC     SQL
SELECT   title, copies INTO :bookname, :nocopies
FROM     books
WHERE    title = 'ISRAEL'
```

אין זו הדרך היחידה להעברת נתונים, אלא הפשוטה שבהן. השיטות להעברת נתונים במהדורה המורחבת של OS/2 תואמות את השיטות במערכות אחרות.

תמיכה סטטית ודינמית

כל משפט SQL עובר הידור לפני הביצוע, ההבדל הוא במועד שפעולה זו נעשית. במצב סטטי, ההידור נעשה פעם אחת בלבד בעת עריכת התכנית אשר משתמשים בה פעמים רבות. במצב דינמי, ההידור נעשה בעת הביצוע של שאילתות מזדמנות אקראיות בלתי מתוכננות מראש. יעילות הביצוע של SQL סטטי גבוהה יותר.

שפה להגדרת נתונים

הגדרת טבלה נעשית בפקודה CREATE TABLE. השמות של השדות בטבלאות השונות יכולים להיות שונים, או שהם יכולים להיות שווים. במקרה האחרון יהיה צורך לרשום את שם הטבלה כמציין לשם השדה. נכתוב למשל ITEM.NUMBER או ORDERS.NUMBER, כאשר "מספר-פריט" הוא NUMBER בשתי טבלאות שונות. לדוגמה הגדרה של טבלת ITEM:

```
CREATE TABLE ITEM    (I_NUMBER       INTEGER NOT NULL,
                      I_DESCRIPTION  CHAR (45))
```

בכל שורה בכל טבלה חייב שדה אחד לפחות להיות שדה-זיהוי או שדה-מפתח, ולכן הוא חייב להכיל ערך כלשהו "לא אפס" (NOT NULL). סוגי נתונים אפשריים הם לדוגמה: CHAR, VARCHAR, INTEGER, DECIMAL, DATE, FLOAT. סוג נתון מיוחד יכול להכיל מחרוזות עד 32000 תווים, והוא מיועד לאחסנת נתונים גרפיים ותמונות ווידאו.

מערכת SQL מאפשרת להוסיף עמודות חדשות לטבלאות קיימות באמצעות הפקודה ALTER TABLE. הרחבת טבלה נעשית באופן דינמי במהלך העבודה, ואפשר להשתמש בה מיד. השדות בעמודות שנוספו הם ריקים (Null). לדוגמה, אם יש להוסיף לטבלת הפריטים עמודה חדשה, אשר מכילה את סוג הפריט (תו אחד אלפנומרי), נרשום:

```
ALTER TABLE ITEM
ADD I_TYPE CHAR (1)
```

מבטלים טבלה בהוראת DROP TABLE.

אפשר ליצור ולבטל אינדקסים, כדי לשפר את ביצועי המערכת. לכן רצוי לבנות אותם עבור שדות (עמודות) שההתייחסות אליהם היא רבה. השימוש באינדקסים הוא אוטומטי.

התוכנה כוללת קטלוג המשמש לשמירת הגדרות הנתונים. הקטלוג בנוי במבנה של טבלאות ועל-כן ניתן לשלוף ממנו מידע באמצעות הוראות המערכת הרגילות. הגדרת הנתונים והטיפול בהם פשוטים ומותאמים למשתמש, שאינו מנהלן בסיס נתונים כמקובל במערכות גדולות.

תכניות שירות ומישקים לתכניות יישומים

התוכנה כוללת פונקציות שהן מישקים לתכניות יישומים (Application Program Interface - API). תכניות השירות מאפשרות ליצור בסיסי נתונים, לבטל, לשלב ולהפריד בסיסי נתונים, לקבל מידע אודות בסיסי נתונים ולשנות פרמטרים הקשורים בהם. תכניות שירות מסוג אחר יכולות לקלוט נתונים לתוך טבלה (IMPORT), או להפיק נתונים מתוך טבלה (EXPORT), לבצע גיבוי, שחזור וארגון מחדש. פונקציית השירות RUNSTATS מיועדת לעדכון הרישומים הסטטיסטיים הדרושים לבקרה ושיפור הביצועים (Performance Optimizer).

הידור מראש וקישור

הידור וקישור מיועדים למקרים שבהם משפטי SQL כלולים בתכנית יישום, כמו

למשל בתכנית הכתובה בשפת C. משפטי SQL אינם משפטים "חוקיים" בשפת C ועל-כן יש צורך לערוך את התכנית המשולבת באמצעות קדם-מהדר, כדי שיהיו בה רק משפטים חוקיים בשפה זו. לאחר מכן התכנית עוברת הידור כדי להפיק תכנית הניתנת לביצוע.

בזמן העריכה של קדם-המהדר נבדקים משפטי SQL ונקראות השגרות המשרתות את הפונקציות הכלולות בהם. באותה עת גם נקבעים מסלולי הגישה לטבלאות כדי להבטיח ביצוע יעיל. הדיון הזה מתייחס להפעלה סטטית של SQL, כי בהפעלה דינמית משתמשים במשפטי SQL בלבד.

שירותי בסיס הנתונים

עיבוד מרובב ושירות למשתמשים רבים

במערכת OS/2-EE ניתן להפעיל מספר תהליכים (processes) בו-זמנית. מנהל בסיס הנתונים מנצל תכונה זו כדי לתמוך בגישה בו-זמנית של מספר תכניות אל אותו בסיס הנתונים. כדי לשמור על עקביות ושלמות בסיס הנתונים בעת ביצוע פעולות/תנועות, הוא מפעיל שירותים של חסימה ושחרור, אשר מונעים עדכון בו-זמני על-ידי משתמשים אחדים.

המשתמשים יכולים לנצל מגוון שירותים זה כדי להשתמש בעת ובעונה אחת במספר בסיסי נתונים. לדוגמה, ניתן להפעיל תכנית דיווח ברקע, ובאותו זמן להפעיל גם תכניות לקליטת תנועות ועדכון, אשר משתמשות כולן באותו בסיס הנתונים.

התמיכה בעיבוד מרובב ושירות למשתמשים רבים הינה הבסיס של השירותים מרחוק (אשר אינם זמינים בעת כתיבת הספר). מרכיב זה של מנהל בסיס הנתונים מאפשר תמיכה בתחנות עבודה רבות הקשורות ב-LAN ומקבלות שירותים משרת (Server) אחד שהם "לקוחותיו".

תמיכה בתנועות

תנועה הינה יחידת עבודה אטומית, כמו למשל עדכון לקובץ, או שחזור המצב הקודם לאחר תקלה. בסיס הנתונים אינו שלם כאשר רק חלק של התנועה מוצג בשדות הרשומה במהלך העדכון. יש צורך לוודא שהפעולה תושלם תמיד, או שהמצב הקודם ישוחזר, כדי שהתהליך לא ישאר במצב של "ביצוע חלקי/לא מושלם".

תנועה נועלת את בסיס הנתונים מפני גישה של תנועה אחרת בשעה שהיא פועלת לעדכון או שחזור. המערכת משתמשת לשם שחזור בקובץ הרישום הכרונולוגי (log file), אשר מציין נקודות בקרה. הוא מאפשר לשחזר מצב מנקודת בקרה כלשהי, בהתאם לדרישת המשתמש, ולאו דווקא מתחילת תהליך העיבוד.

גישה בו-זמנית

מנהל בסיס הנתונים מאפשר לתהליכים רבים גישה בו-זמנית (cuncurrent access) אל בסיס הנתונים. המערכת משתמשת במנגנון נעילה, באמצעות

פקודות נעילה המופקות באופן אוטומטי בעת העדכון, או במהלך העבודה, בפקודה LOCK TABLE. מנגנון השליטה של מנהל בסיס הנתונים משתדל לנעול פחות ככל האפשר, כדי לאפשר יותר זמן חופשי לשירות של המשתמשים ולמנוע היווצרות תור.

נעילה יכולה להתבצע ברמה של טבלה, או ברמה של רשומה בודדת. העדיפות היא כמובן לנעילה לפי רשומה, כדי לאפשר גישה של משתמשים אחרים לרשומות אחרות בטבלה. ניתן לבצע נעילה מוחלטת, או נעילה שתאפשר קריאה בלבד, אך לא כתיבה. משתמשים שונים יכולים לגשת אל האינדקס באותו זמן.

במערכת פעילה ודינמית יכולים להיווצר מצבים של נעילה ללא מוצא (dead locks), אשר יכולים לגרום לחסימה והפסקת פעולה. מצב חמור זה נמנע באופן אוטומטי על-ידי מנהל בסיס הנתונים, אשר יודע לגלות מצבים אלה ולהשתחרר מהם כדי להמשיך את הפעילות התקינה.

הידור ואופטימיזציה

משפטי SQL סטטיים עוברים הידור וקישור. אפשר לעשות זאת ל"תכנית" SQL עצמאית, אך חייבים לעשות זאת כאשר משפטי SQL משולבים בתכנית אחרת, כגון תכנית בשפת C. מהלכי ההידור והקישור דומים לאלה שנעשים בכל שפת תכנות שהיא. תכנות מראש מכוון לביצוע שאילתות קבועות, אשר יכולות לקבל בכל הרצה פרמטרים שונים מן המשתמש, כמו למשל טווח ערכים לחיפוש, קריטריון השוואה שונה לאחד הערכים וכו'.

רכיב האופטימיזציה של מנהל בסיס הנתונים בוחן את משפטי SQL המרכיבים את השאילתה ובודק אם קיים אינדקס, מהן הטבלאות שיש לחפש בהן ומה גודלן ועוד. על-פי נתונים אלה ועל-פי אלגוריתם הבנוי בו, נקבע מסלול הגישה (Access Path) לחיפוש הנתונים הדרושים, ובכלל זה סדר החיפוש בטבלאות והשימוש באינדקס.

ניהול האחסנה

שירותי מנהל בסיס הנתונים מנצלים את מערכת הקבצים האינטגרלית של OS/2, כדי להקים ולאחסן את הטבלאות המרכיבות את בסיס הנתונים. כל טבלה נשמרת בקובץ נפרד. כל קובץ מחולק פנימית לדפים בני 4K בתים כל אחד, המהווים יחידות קריאה/כתיבה של OS/2. חלוקה זו הינה לצורך פעולות קלט/פלט ואינה קשורה למבנה הלוגי של הקבצים, אשר יכולים להחזיק רשומות שאורכן עד 32,000 בתים.

לכל בסיס נתונים פעיל מוקצה מאגר פעולה (buffer) בזיכרון, אשר גודלו לפחות 4KB. מאגר זה מנוהל כזיכרון מטמון (cache), שבו נוהל הפעילות הוא LRU (Least Recently Used). כלומר, כאשר צריך לקרוא דף מן הדיסק, משתמשים בשטח המאגר שבו נמצא הדף "הזקן" ביותר, או הדף שנעשה בו שימוש בזמן הרחוק ביותר. דפים של אינדקסים ודפים דרושים (כפי שנקבע), נמצאים כל הזמן במאגר ואינם מוחלפים.

בעת כתיבה לקובץ, הרשומות נאגרות במאגר ורק כאשר יש דף שלם הוא נרשם בדיסק. הרשומות הנשלחות אל קובץ הרישום הכרונולוגי נרשמות תחילה אל הדיסק, לפני שהדף השלם המכיל אותן נכתב לדיסק.

אבטחה

מנהל בסיס הנתונים מספק אפשרות לסיסמאות גישה ברמה של בסיסי הנתונים. כלומר, משתמש מורשה לטפל בבסיס הנתונים רק לאחר שהוא מוסר למערכת את סיסמת הגישה שלו. הסיסמה נקבעת בעת יצירת הקובץ, וניתן לעדכן אותה. האחראי למערכת יכול לציין שאינו רוצה בסיסמאות להרשאת הגישה.

תצפית הנתונים VIEW, הינה אמצעי נוסף חשוב לאבטחת נתונים על-ידי מניעת גישה אל חלקי טבלאות.

תמיכה בשפות לאומיות

התמיכה של מנהל בסיס הנתונים בשפות לאומיות מתאפשר על-ידי שמירה של ההנחיות למשתמש, הכוללות הודעות ותפריטים הנמצאים בקבצים נפרדים. הפרדה זו מאפשרת לתרגם נתונים אלה בהתאם לדרישות של כל שפה. התאמות נוספות לדרישות של ארצות שונות קשורות לצורת ההצגה של תאריך וזמן, סדר מיון של נתונים (א"ב, ספרות וסימנים), סימון מספר עשרוני ועוד.

שירותים למרחק

שירותים אלה (אינם זמינים עדיין) תומכים בעיבוד של בסיסי נתונים מבוזרים ברשתות (LAN). הם מאפשרים למשתמש להוציא הוראת SQL לבסיס נתונים שאינו נמצא במחשב שלו, אלא במחשב אחר הקשור לרשת.

כדי לאפשר שירות זה, יש רישום של בסיסי הנתונים השונים בקטלוג המערכת. נערכת בדיקה בקטלוג היכן נמצא הקובץ המכיל את הנתונים הדרושים. אם הקובץ נמצא במחשב מרוחק, מעבירים אליו את ההוראות של SQL באמצעות רכיב APPC של מנהל התקשורת ב-OS/2. ההוראות מתבצעות במחשב המרוחק והתוצאה של העיבוד מוחזרת אל התכנית המבקשת, כאילו העיבוד נעשה בפיקוח ישיר שלה במחשב שבו היא נמצאת. פעילות זו שקופה לתכנית היישום ולמשתמש. נוהלי אבטחת המידע נשמרים במערך המבוזר, כמו במחשב יחיד.

סיכום

תוכנת מנהל בסיס הנתונים בגירסת OS/2-EE משתמשת בטכנולוגיות ייחודיות רבות של יבמ. היא פותחה במקביל לפיתוח מערכת ההפעלה OS/2, ולפיכך היא מותאמת לעבוד בסביבה זו ומנצלת את כל האפשרויות שלה.

התוכנה מספקת ביצועים משופרים ופונקציות לאבטחת שלמות נתונים (data integrity). היא תואמת בפונקציות שלה לתוכנות יבמ האחרות לניהול בסיסי נתונים טבלאיים, ובכך היא תאפשר עבודה במערכת מבוזרת.

תוכנת מנהל בסיס הנתונים והפונקציות הכלולות ב-OS/2-EE הינם תשתית לפיתוח יישומים מקיפים בתחומים שונים על-ידי משתמשי המערכת.

נספח ג

מערכת אישית יבמ/2 – PS/2

מבוא

מערכת אישית יבמ/2 היא משפחה של מחשבים אישיים בארכיטקטורה חדשה, אשר מהווה את הצעד הראשון במסלול גידול ארוך טווח. ארכיטקטורה זו מבטיחה את עקרון העתיד במיחשוב: אחידות פיתוח ושימוש ביישומים לכל אורך קו המוצרים של יבמ.

המערכת האישית PS/2 הינה בעלת תכונות טכניות משופרות לעומת המחשב האישי, במהירות עיבוד וביכולת אחסון נתונים. כושר ביצוע של 5.2 מיליון פקודות בשניה, 16 מיליון בתים של זיכרון מרכזי, 628 מיליון בתים בדיסקים קשיחים ובתוספת של עד 1.6 מיליארד בתים בתקליט אופטי חדיש, עושים את הדגם הגדול במשפחה לאחד המחשבים האישיים החזקים בעולם.

המערכת האישית יבמ/2 מהווה המשך ושכלול של המחשב האישי של יבמ. היא מאפשרת למשתמש לנצל את השקעותיו הנוכחיות בציוד ובתוכנה, להמשיך להשתמש בתוכנה הקיימת, להפעיל יחידות חדשות בצד יחידות קיימות ולהעביר מידע בין מחשבים משני הסוגים.

הטכנולוגיה המתקדמת של המערכת כוללת חלק ניכר מהאביזרים הדרושים לפעילויות שונות, כחלק תקני ממנה. כרטיסים ואביזרים לתקשורת, למדפסת, למצגים ולעכבר בנויים על "לוח האם" של כל מחשב ועוד נותר מקום לכרטיסים אחרים, לצורך גידול בעתיד. כל הדגמים של המערכת האישית יבמ/2, הם בעלי עוצמה גדולה בהרבה מהמחשבים האישיים הקיימים וקטנים מהם במימדים.

ארכיטקטורה וטכנולוגיה חדישות

המערכת האישית יבמ/2 בנויה בארכיטקטורה ייחודית האופיינית למחשבים בינוניים וגדולים של יבמ, Micro Channel Architecture, אשר מאפשרת ביצוע של מספר משימות קלט/פלט במקביל. מבנה זה מעניק למחשבי המשפחה עוצמה רבה ואפשרויות גידול נרחבות (פרט לדגמים 25 ו-30).

כל דגמי המערכת בנויים בטכנולוגיה חדישה של מיזעור ואינטגרציה, ובחלק מהם אף שולבו רכיבים זהים לאלה שבמחשבי הענק של יבמ. חידושים אלה איפשרו שיפור רב ביחס עלות/ביצוע של המערכת האישית יבמ/2 ופתחו אפשרויות חדשות רבות בפני המעוניינים לפתח יישומים חדשים, לעסוק בעבודות הדורשות משאבי מחשב רבים, לשלב מחשבים ברשתות תקשורת ועוד.

על גבי המצגים החדשים אפשר לקבל תמונות ותמלילים חדים וברורים, באיכות דומה לתמונות ודפים מודפסים. מגוון הצבעים שניתן להציג בו-זמנית רחב

ביותר ומגיע ל-256 מתוך 262,144 אפשרויות!

תקשורת רב תכליתית

גורם חשוב הוא הפתיחות: פתיחות לחידושים טכנולוגיים ולתקשורת מסוגים שונים. המערכת האישית יבמ/2 מתקשרת למחשבים מרכזיים גדולים יותר, למחשבים אישיים קיימים, והיא תאפשר להתקשר עם מחשבי העתיד של יבמ ועם מקורות מידע אחרים.

אין אלה מחשבים ליישום בודד, אלא מערכות אישיות מקיפות, שניתן לחברן זו עם זו, להשתמש בהן לצרכים רבים ולבצע בעזרתן משימות רבות בו-זמנית.

עברית

יבמ פיתחה עבור המערכת החדשה תמיכה מקיפה בעברית, התואמת את התמיכה העברית במחשבים האישיים של יבמ. היא מאפשרת למשתמשים בישראל להנות מהיתרונות הנובעים מתכונותיה של המערכת החדשה.

מאפיינים ושימושים עיקריים

דגמים 25, 30

מחשבים שולחניים קטנים לשימוש כללי, אשר נועדו למיחשוב אישי, לכתת לימוד, לעסק קטן, כתחנת עבודה בודדת או משולבת ברשת. בדגם 25 המצג בנוי כחלק מן היחידה.

שימושים:

- תחנת עבודה עצמאית ליישומים נפוצים, כמו גיליון עבודה אלקטרוני, עיבוד תמלילים, גרפיקה, הנהלת חשבונות, לומדה וכו'.
- תחנת עבודה ברשת תקשורת מקומית לביצוע יישומים עצמאיים. יישומים מחלקתיים (למשל, דואר אלקטרוני) ויישומים עסקיים הדורשים שטחי אחסון גדולים יותר.
- תחנת עבודה אינטליגנטית בתקשורת למחשב מארח.
- תחנת עבודה משופרת להפעלת מתאמים ויחידות קיימות ממשפחת ה-PC.

המחשבים מתאימים למי שמעוניינים בכניסה בעלות נמוכה לתחום המיחשוב האישי.

דגם 50

מחשב שולחני בעל עוצמת חישוב רבה להפעלת יישומים מתוחכמים. דגם זה עוצב בארכיטקטורה החדישה של Micro channel.

שימושים:
- תחנת עבודה עצמאית ליישומים הדורשים כוח חישוב ופונקציות מתקדמות: ריבוי משימות, ניהול חלונות, גיליונות עבודה אלקטרוניים גדולים, עיבוד תמלילים ושימוש משרד אחרים, וכן תכניות חישובים מדעיים והנדסיים.
- תחנת עבודה ברשת תקשורת מקומית, או למחשב מרכזי, לשילוב יישומים עצמאיים עם אמצעי אחסון גדולים, תקשורת מורכבת ויישומים מחלקתיים רבי-משתמשים. כוח החישוב והגרפיקה של המחשב בשילוב עם אמצעי אחסון שברשת תקשורת מאפשרים הפעלת יישומי תיב"מ.

המחשב מתאים לכל המעוניין בתחנה עצמאית בעלת עוצמת חישוב רבה, או במחשב חזק כמרכיב ברשת תקשורת.

דגם 60

מחשב בעל עוצמת חישוב ואמצעי אחסון רבים להפעלת יישומים מתוחכמים ועיבוד כמויות גדולות של נתונים. מעוצב בארכיטקטורת Micro channel, ונועד להצבה על הרצפה.

שימושים:
- תחנת עבודה עצמאית, או משולבת ברשתות תקשורת מורכבות, לביצוע מגוון רחב של פונקציות ויישומים: ריבוי משימות, ניהול חלונות, עבודה על גיליונות עבודה אלקטרוניים ומאגרי נתונים גדולים, יישומים גרפיים מסוג תיב"מ, עיבודים מדעיים והנדסיים ועיבודים מסחריים עסקיים הדרושים משאבים רבים.
- שרת ברשתות תקשורת מקומיות קטנות ובינוניות.

המחשב מתאים לעסקים הדורשים עוצמת עיבוד, אמצעי אחסון ואפשרויות גידול נרחבות.

דגמים 70, 80

מחשבים רבי-עוצמה ורבי-משאבים, אשר עוצבו בארכיטקטורה של Micro channel. דגם 70 הוא מחשב שולחני, ואילו דגם 80 נועד להצבה על הרצפה.

שימושים:
- תחנת עבודה עצמאית, או משולבת ברשתות תקשורת מורכבות, לביצוע מגוון רחב של פונקציות ויישומים: ריבוי משימות, ניהול חלונות, מאגרי נתונים גדולים, יישומי תיב"מ מורכבים, עיבודים מדעיים והנדסיים ועיבודים עסקיים בהיקף גדול. למודל 80 אפשרויות רבות יותר לאחסנת נתונים.
- שרת ברשתות תקשורת מקומיות בינוניות וגדולות.

המחשבים מתאימים לעסקים הדורשים עוצמת עיבוד ואמצעי אחסון לטיפול במשימות רבות היקף ואפשרויות גידול נרחבות.

IBM PS/2 - נתונים טכניים

	• דגם 25	דגם 30			דגם 50		דגם 60	
		30-002	30-021	30-H21	50-021	50-061	60-041	60-071
מעבד (אינטל)	8086	8086	8086	80286	80286	80286	80286	80286
מהירות (MHz)	8	8	8	10	10	10	10	10
מעבד מתימטי	8087	8087	8087	80287	80287	80287	80287	80287
מערכות הפעלה	PC DOS 3.3	PC DOS 3.3	PC DOS 3.3	PC DOS 3.3 OS/2	PC DOS 3.3 OS/2	PC DOS 3.3 OS/2	PC DOS 3.3 OS/2	PC DOS 3.3 OS/2
זיכרון ROM (KB)	64	64	64	128	128	128	128	128
זיכרון RAM מוכלל (MB)	0.512-0.64	0.64	0.64	1	1	1	1	1
זיכרון RAM מכסימלי על לוח המערכת (MB)	0.64	0.64	0.64	4	1	2	1	1
זיכרון RAM מכסימלי (MB)	0.64	2.64	2.64	16	16 ••	16	16	16
מהירות גישה לזיכרון nsec	150	150	150	120	150	85	150	150
כונן תקליטון 3.5" (MB)	0.72	0.72 2 כוננים	0.72	1.44	1.44	1.44	1.44	1.44
תקליט קשיח (MB)	-	-	20	20	20	60	44	70
תקליט קשיח נוסף (MB)	20	-	-	-	60 •• החלפה	-	44	70, 115
כניסות קלס/פלט	2	3	3	3	3	3	7	7

	דגם 70			דגם 80			
	70-F61	70-121	70-A21•••	80-041	80-071	80-111	80-311
מעבד (אינטל)	80386	80386	80386	80386	80386	80386	80386
מהירות (MHz)	16	20	25	16	16	20	20
מעבד מתימטי	80387	80387	80387	80387	80387	80387	80387
מערכות הפעלה	PC DOS 3.3 OS/2, AIX	PC DOS 3.3 OS/2, AIX	PC DOS 3.3 OS/2, AIX	PC DOS 3.3 OS/2, AIX	PC DOS 3.3 OS/2, AIX	PC DOS 3.3 OS/2, AIX	PC DOS 3.3 OS/2, AIX
זיכרון ROM (KB)	128	128	128	128	128	128	128
זיכרון RAM מוכלל (MB)	2	2	2	1	2	2	2
זיכרון RAM מכסימלי על לוח המערכת (MB)	6	6	8	2	2	4	4
זיכרון RAM מכסימלי (MB)	16	16	16	16	16	16	16
מהירות גישה לזיכרון nsec	85	85	80	80	80	80	80
כונן תקליטון 3.5" (MB)	1.44	1.44	1.44	1.44	1.44	1.44	1.44
תקליט קשיח (MB)	60	120	120	44	70	115	314
תקליט קשיח נוסף (MB)	-	-	-	44	70, 115 314 ••	70, 115 314 ••	70, 115 314
כניסות קלס/פלט	2 של 32 סיביות 1 של 16 סיביות			3 של 32 סיביות 4 של 16 סיביות			

• PS/2 M25 מופיע במספר דגמים.
•• שיפור בדגם קיים.
••• עם מעבד מיוחד - 82385 וזיכרון מטמון (CHACHE MEMORY) בגודל 64KB.

יחידות היקפיות

מצגים

משפחת המערכת האישית יבמ/2 כוללת חמישה מצגים גרפיים המאפשרים הצגה באיכות גבוהה של טקסט, גרפיקה ותמונות. המצגים מבוקרים על-ידי מתאם הבנוי בתוך יחידת המערכת.

מצג יבמ 8503 הוא מצג גרפי חד-צבע עם מרקע בגודל "12, עליו ניתן להציג מידע ב-64 גוונים של אפור. מצג אחר תואם, הוא בגודל "9.

מצגים יבמ 8512, יבמ 8513 ו-יבמ 8514 הינם מצגים צבעוניים גרפיים בעלי מרקעים "12, "14 ו-"16 בהתאמה, שניתן להציג בהם טקסט ותמונות ב-256 צבעים מתוך מבחר של 262 אלף צבעים.

תקשורת

למחשבי המערכת האישית אפשרויות רבות לשילוב ברשתות תקשורת מרכזיות או מקומיות.

- בתחום התקשורת למחשב מרכזי מארח - מתאמים שונים מרחיבים את אפשרויות החיבור: מתאם אסינכרוני במהירות 19.2Kbs המוכלל בכל אחד מדגמי המערכת. מתאם בעל שתי יציאות לתקשורת אסינכרונית ומתאם המשלב בכרטיס אחד את פרוטוקולי התקשורת: אסינכרוני, BSC, SDLC ו-HDLC.
- בתחום רשתות התקשורת המקומיות - מתאמים ותוכנה המאפשרים בניית רשתות במגוון פונקציות:
  - רשת פס-בסיס (Baseband) - לכניסה נוחה ובעלות נמוכה לתחום הרשתות המקומיות.
  - רשת מקומית (PC Network) - לשילוב סוגי אותות נוספים (אודיו, וידאו..) במקביל לתקשורת מחשבים.
  - רשת טבעת הסמן (Token Ring) - בעלת קיבולת העברת נתונים גבוהה ואפשרות שילוב מחשבים מסוגים שונים.

מוצרי התקשורת האלה מאפשרים שילוב ברשת אחת של מחשבים אישיים, מחשבי מערכת אישית/2, וחיבור רשתות מקומיות למחשבים מארחים באמצעות Gateway.

יחידות נלוות

למערכת האישית מגוון של יחידות נלוות. הבולטות ביניהן:

- כונן תקליט אופטי - כונן יבמ 3363 המאחסן נתונים על-גבי תקליטים אופטיים ניידים שתכולת כל אחד 200Mb. התקליט האופטי מיועד לשימוש ביישומים הדורשים כמות גדולה במיוחד של נתונים וכשיש צורך להעביר נתונים ממקום למקום, או לשמרם במקום נפרד מסיבות בטיחות.

- אביזרי קישור - מטרת אביזרים אלה לאפשר העברת נתונים ותכניות בין המערכת האישית למחשבים האישיים של יבמ. האביזרים כוללים כונני תקליטונים "5.25, "3.5 ומתאם לחיבור ישיר בין מחשבים משתי המשפחות.

מדפסות

המדפסות של מערכות המחשב האישי מתאימות גם למערכת יבמ/2. הן מאפשרות למשתמש לבחור במדפסת המתאימה לצרכיו הייחודיים: הדפסת טקסט ותמונות באיכות גבוהה, מהירות הדפסה, צורות הזנת נייר שונות ועוד.

- IBM proprinter II - 4201: מרכב צר, פונקציות משופרות.
- IBM proprinter X24 - 4207: מרכב צר, ראש הדפסה בן 24 סיכות להדפסות איכות וגרפיקה ברמה גבוהה.
- IBM proprinter XL24 - 4208: מרכב רחב, ראש הדפסה בן 24 סיכות להדפסת איכות, גרפיקה ברמה גבוהה.
- IBM quietwriter III - 5202: מדפסת איכות חרישית. תהליך ההדפסה נעשה בטכנולוגיה חדשנית המבוססת על השחרת הנייר באמצעות חימום סרט ההדפסה. מהירות ההדפסה עד 274 תווים לשנייה, גרפיקה ברזולוציה של עד 240*240 נקודות לאינץ'.
- IBM pageprinter - 4216 - מדפסת דפים שולחנית בטכנולוגיית לייזר. קצב הדפסה מירבי של 6 דפים לדקה ורזולוציה של 300*300 נקודות לאינץ'. המדפסת תואמת לסטנדרט postscript ומתאימה גם ליישומי הוצאה לאור שולחנית (DTP).

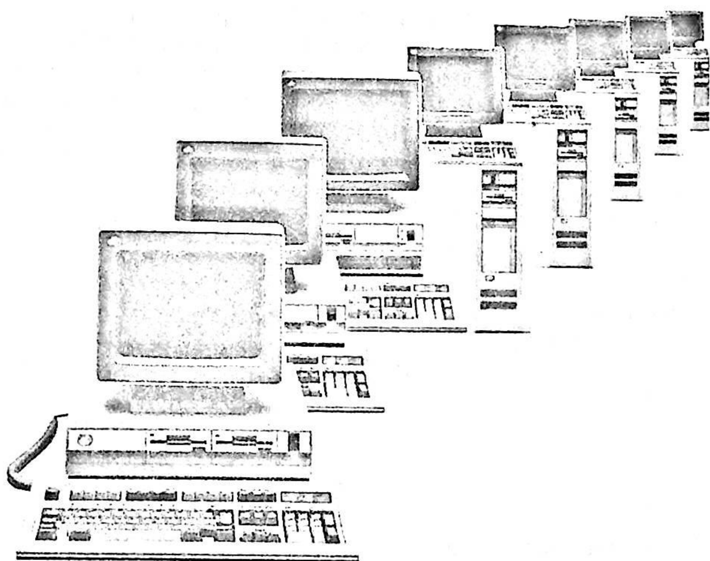
תוכנה

למערכת האישית מגוון רחב של תוכנה המביאה לידי ביטוי את תכונות החומרה המתקדמות שלה.
ניתן להפעיל במערכת שלוש מערכות הפעלה:

PC DOS 3.3 - (או גירסה מתקדמת יותר) תומכת בכל דגמי המחשב האישי ובמערכת האישית. גירסה זו תואמת את הפונקציות של PC DOS 3.2, מאפשרת שיפורים בביצועים של פעולות קלט/פלט ומוסיפה פקודות חדשות. PC DOS 3.3 מאפשרת להפעיל במערכת האישית יבמ/2 את המגוון הגדול של תוכנה למחשבים אישיים של יבמ, תוך ניצול הביצועים המשופרים והאפשרויות הגלומות בה.

OS/2 (Operating System/2) - מערכת חדשנית, אשר נועדה לנצל את הביצועים המתקדמים של המעבדים אינטל 80286 ו-80386, תוך שמירה על תאימות למגוון התוכנות שפותחו עבור מערכת ההפעלה PC DOS. מערכת זו מאפשרת ריבוי משימות (multi-tasking), ריבוי תכניות (multi-programming), ניצול מרחב זיכרון של עד 16Mb, בסיס נתונים טבלאי, יישומי תקשורת מוכללים והשתלבות בארכיטקטורת יישומים בין-מערכתית של יבמ (SAA).

AIX PS/2 (Advanced Interactive Executive) - מערכת הפעלה רבת משתמשים (multi-user) לדגמים 70, 80. מערכת ההפעלה תומכת בזיכרון בפועל וב-16 משתמשים בו-זמנית. היא מספקת סביבת עבודה מבוססת UNIX במגוון רחב של תחנות עבודה.



אינדקס

האינדקס ערוך במיון של המונחים באנגלית והוא כולל תרגום, או הסבר קצר של המונח, כפי שהשתמשנו בו בספר.

מראי המקום מציינים את המקומות החשובים שבהם הסברנו את המונח או שדנו בו בהקשר כלשהו. לרוב ציינו את טווח העמודים, ולעתים רק את העמוד המציין את תחילת הטיפול בנושא. הפירוט של מראי המקום ניתן תחת המונח בשמו המלא, ולעתים תחת הצורה המקובלת של ראשי התיבות (כמו, I/O). במקרים רבים ניתן פירוט המונחים תחת כותרת הנושא (כמו למשל, Memory, או Device Driver). פקודות, מישקי תכנות ופרמטרים ניתנו תחת שם הנושא (כמו למשל פקודות API), אם כי לפעמים רשמנו אותם גם לפי הסדר המילוני.

אנו מקווים שהשימוש באינדקס, במשולב עם תוכן העניינים המפורט, יסייע לך בלימוד ובעיון בספר זה.

מישקים לתכנות יישומים (API)

מערכת ההפעלה OS/2 יועדה על ידי מתכנניה לשנות את פני המיחשוב האישי. בעזרת ספר זה תיווכח שהיא עתירת פונקציות וקלה יותר לשימוש מאשר DOS, גם תלמד כיצד תוכל לרתום לשירותך את עוצמת התכנות חסרת התקדים שהיא מאפשרת. לפניך מדריך להכרה מעמיקה של OS/2, אשר יאפשר לך להשיג את היתרונות שבהכרת מערכת ההפעלה המתקדמת, על עוצמתה, תכונותיה ואפשרותיה המגוונות.

בפרק הראשון ניתנת **סקירה מקיפה על המאפיינים, הפונקציות והיישומים,** אשר מייחדים את OS/2 בתחום המיחשוב האישי. תלמד כיצד התאפשר התפעול של OS/2 בעזרת **המיקרו–מעבד 80286,** ליבה של מערכת החומרה. בהמשך אנו דנים בהרחבה בנושאים הבאים:

- **סביבה של ריבוי משימות** (Multitasking) **וישומים:** מה זה, איך זה מתפקד, איך צריך וניתן לנצל תכונות אלו ביעילות בעת הפעלה של תחנת עבודה אינטליגנטית בפיקוח של OS/2 בלבד, או בשילוב DOS.

- **עקרונות הפעולה של** OS/2: ניהול זכרון ותמיכה בזכרון יתר, קישור דינמי בעת טעינה ובעת ריצה, טעינת סגמנטים, הפעלת יחידות קלט/פלט, ניהול התקנים מבוקרי–פסקים, ביצוע של יישומים במקביל, תקשורת בין תהליכי עיבוד, שקולי תפעול, תצורות, תמיכה בשפות לאומיות, ועוד.

- **מנהל התצוגה** (Presentation Manager): אוסף השירותים לשם פיתוח והרצה של יישומים בצורה הידברותית במסכי תצוגה. עקרונות תכנות באמצעות מנהל התצוגה והדגמת תהליך לבניית ישום בשיטות מתקדמות.

- **מנהל בסיס הנתונים** (Database Manager): ניהול של מאגרי נתונים בשיטת בסיס הנתונים הטבלאי (Relational DB). תיאור של שפת SQL לניהול הנתונים, ביצוע שאילתות והפקת דו"חות, אשר תואמת למערכות בסיסי הנתונים האחרות של יבמ.

- **ועוד** – דוגמאות תכנות מוסברות, אשר יאפשרו למי שמעוניין בכך, ללמוד כיצד לפתח מערכי תוכנה מתקדמים ב–OS/2 וכיצד לנצל כראוי את תכונותיה ואת עוצמתה. מידע על המאפיינים והשימושים העיקרים של המערכות האישיות יבמ/2 (PS/2) – התשתית להשגת היתרונות של OS/2.

הספר מיועד למי שמעוניין להכיר את מערכת ההפעלה והשירותים שהיא מספקת למשתמש בה, עבור המתמחים במערכות מיחשוב אישי, מנתחי מערכות, מתכננים ומפתחי יישומים. הספר פותח בפניהם עולם חדש ומופלא, רב מימדים ואתגרים, באמצעות הצגה ותיאור של הפונקציות המתקדמות ביותר של OS/2.

מסת"ב ISBN 956-361-004-X